2007-06

A dynamic three-dimensional network visualization program for integration into cyberciege and other network visualization scenarios

Sledz, Daniel A.; Coomes, Donald E.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

A DYNAMIC THREE-DIMENSIONAL NETWORK VISUALIZATION PROGRAM FOR INTEGRATION INTO CYBERCIEGE AND OTHER NETWORK VISUALIZATION SCENARIOS

by

Daniel A. Sledz
Donald E. Coomes

June 2007

Thesis Advisor: Mathias Kölsch
Second Reader: Michael Thompson

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** June 2007	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE A Dynamic Three-Dimensional Network Visualization Program for Integration into CyberCIEGE and Other Network Visualization Scenarios		**5. FUNDING NUMBERS**	
6. AUTHORS Daniel A. Sledz, Donald E. Coomes			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE** A	

13. ABSTRACT (maximum 200 words)

Detailed information and intellectual understanding of a network's topology and vulnerabilities is invaluable to better securing computer networks. Network protocol analyzers and intrusion detection systems can provide this additional information. In particular, game-based trainers, such as CyberCIEGE, have been shown to improve the level of training and understanding of network security professionals. This thesis' objective is to enhance these applications by developing NTAV3D, or, Network Topology and Attack Visualizer (Three Dimensional).

NTAV3D is a tool that displays network topology, vulnerabilities, and attacks in an interactive, three dimensional environment. This augments the design and gameplay of CyberCIEGE by increasing gameplayer interaction and data display. Additionally, NTAV3D can be expanded to provide this capability to network analysis and intrusion detection tools. Furthermore, NTAV3D expands on ideas and results from related work of the best ways to visualize network topology, vulnerabilities, and attacks.

NTAV3D was created using open-source software technologies including Xj3D, X3D, Java, and XML. It is also one of the first applications to be built with only the Xj3D toolkit. Therefore, the development process allowed evaluation of these technologies, resulting in recommendations for future improvements.

14. SUBJECT TERMS Network visualization, Networking, Networks, CyberCIEGE, Network Security, Intrusion Detection Systems, Extensible Markup Language, XML, Java, Xj3D, X3D			**15. NUMBER OF PAGES** 135
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

A DYNAMIC THREE-DIMENSIONAL NETWORK VISUALIZATION PROGRAM FOR INTEGRATION INTO CYBERCIEGE AND OTHER NETWORK VISUALIZATION SCENARIOS

Daniel A. Sledz
Ensign, United States Navy
B.S., Purdue University, 2006

Donald E. Coomes
Ensign, United States Navy
B.S., Oregon State University, 2006

Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE IN MODELING,
VIRTUAL ENVIRONMENTS, AND SIMULATION (MOVES)**

from the

**NAVAL POSTGRADUATE SCHOOL
June 2007**

Authors:	Daniel A. Sledz
	Donald E. Coomes
Approved by:	Mathias Kölsch Thesis Advisor
Approved by:	Michael Thompson Second Reader
Approved by:	Mathias Kölsch Chair, MOVES Academic Committee

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Detailed information and intellectual understanding of a network's topology and vulnerabilities is invaluable to better securing computer networks. Network protocol analyzers and intrusion detection systems can provide this additional information. In particular, game-based trainers, such as CyberCIEGE, have been shown to improve the level of training and understanding of network security professionals. This thesis' objective is to enhance these applications by developing NTAV3D, or, Network Topology and Attack Visualizer (Three Dimensional).

NTAV3D is a tool that displays network topology, vulnerabilities, and attacks in an interactive, three dimensional environment. This augments the design and gameplay of CyberCIEGE by increasing gameplayer interaction and data display. Additionally, NTAV3D can be expanded to provide this capability to network analysis and intrusion detection tools. Furthermore, NTAV3D expands on ideas and results from related work of the best ways to visualize network topology, vulnerabilities, and attacks.

NTAV3D was created using open-source software technologies including Xj3D, X3D, Java, and XML. It is also one of the first applications to be built with only the Xj3D toolkit. Therefore, the development process allowed evaluation of these technologies, resulting in recommendations for future improvements.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

First and foremost, we would like to thank our families for their love and support during the long hours of work that went into this thesis.

We would also like to thank Dr. Mathias Kölsch and Michael Thompson, for their guidance during the creation of this work. They greatly focused and guided both our writing, and programming to ensure that the final product was one we all could be proud of.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

Computer networks are the interconnections of one or more computers for the purposes of communicating and sharing resources. Often, these networks can connect a myriad of disparate hardware systems. Thus, securing and managing these networks can become a daunting task. This is especially true for an inexperienced network administrator. One way this issue is addressed is increased training for both the novice and expert network manager.

However, part of the difficulty in network management and security in general involves being able to visualize the network. This concept of network visualization involves such aspects as how a network's components are arranged, otherwise known as the network's topology, as well as its potential vulnerabilities. This is a complex subject, and as such, it can become difficult to conceptualize a network's level of security. By gaining a better understanding of a network's topology and vulnerabilities, it is hoped that both the networking student, and professional network administrator can gain a better awareness for the networks they are trying to secure.

1. CyberCIEGE

One effort to provide enhanced, effective training is the information assurance training tool CyberCIEGE. This is a computer game-based trainer developed by collaboration between The Center for Information Systems Security Studies and Research (CISR) at the Naval Postgraduate School (NPS) and Rivermind, Inc. Its development was sponsored by the Naval Education and Training Command, the Office of Naval Research, and the Office of the Secretary of Defense. Through this tool, network security and management can be taught in a more stimulating and interactive way. The goals of the game are clear from its initial title video. It poses the question, "Can you keep the network alive?" The program provides players with the ability to setup and defend networks, and to see a visual representation of the consequences of the decisions they make both before and after attacks on the network.

2. X3D: Extensible 3D Graphics

X3D, or Extensible 3D Graphics, is an open-source, royalty free file format and run-time architecture that is used to display three dimensional graphics (What Is X3D?, May 2007). X3D is also an ISO (International Organization for Standardization)-ratified standard, which means it has industry backing as an accepted format for displaying three dimensional graphics. X3D is based on the Extensible Markup Language, or XML, which makes it well suited for programmatic parsing (see Chapter III for more information on XML).

3. Xj3D

Xj3D is the Java-based application programming interface (API) that was used to develop the software product of this thesis. The Xj3D project exists as both a programming toolkit, and as a stand-alone browser that can load and manipulate the previously mentioned X3D or VRML (Virtual Reality Modeling Language) graphics files (Xj3D Project, 2006). For the purposes of this thesis, the Xj3D API, known as the SAI, or Scene Access Interface, is used to create an X3D scene graph entirely within Java code. The Xj3D browser is then programmatically implemented as an application that can be embedded within CyberCIEGE, or launched separately to display the scene graph produced. Additionally, for those only familiar with implementing X3D in its typical stand-alone file format, Appendix A provides a list of the X3D node types that are programmatically implemented in the thesis software product.

4. Network Tools

Many network tools exist that administrators and students can utilize to study, analyze, and defend computer networks. One such group of software is referred to as network protocol analyzers, or "packet sniffers." An example of a very popular packet sniffer is Wireshark (formerly known as Ethereal). This application can be downloaded at http://www.wireshark.org (accessed May 2007). There also exists software that can help network administrators detect intrusions or network attacks that may be occurring in a network. A software package of this type is often known as an Intrusion Detection

Systems, or IDS. One example of such a program is Snort, available at http://www.snort.org (accessed May 2007). Chapter II provides additional information about network tools.

B. WHY A NETWORK VISUALIZATION TOOL IS NEEDED

CyberCIEGE presents a three dimensional virtual office or military command environment that allows players to see users, their computers, and the office layout. However, this "office view" does not depict network connections. The network topology is viewed in a separate screen, and the current network view in CyberCIEGE is a two dimensional view. The placement of components in this view does not correspond to their actual placement in the office view of the game. An additional problem arises with respect to players' ability to visualize network attacks. Currently, a short video clip is played after network attacks occur that explains the various types of attacks and how they evolve. This video clip is generalized to cover all attacks and is not specific to the attack that occurred or the scenario being played.

Additionally, many of the aforementioned network analysis and intrusion detection systems that can be downloaded from the web, such as Snort or Wireshark, have limited or no integrated visualization capabilities. These packages are not able to create a visual display (let alone a three dimensional one) of the network topology based on the information obtained from packets traveling in the networks. Additionally, visual representation of packet paths is not currently implemented in these packages. Thus, when a network security issues arises, it can become difficult to conceptualize the networks nodes affected, as well as the path the attack itself is taking through the network.

C. GOALS AND PROPOSED SOLUTIONS

1. Goals

a. CyberCIEGE Enhancement

The motivation for this thesis stems from the enhancement of CyberCIEGE as a network security trainer. This will be accomplished by augmenting the simple two-dimensional representation of network topology currently implemented in the

game with an interactive, three dimensional visualization tool, allowing game players to see the networks they create from any angle. It is hoped that by providing players with an interactive, three-dimensional visualization, they will be able to gain a better sense of network interconnections created in the scenario being played, and a better understanding of information flow involved in network attacks. This kind of information can be invaluable in learning how to best setup and defend computer networks from attack.

b. Intrusion Detection Tool Enhancement

In addition to providing an enhanced visualization application to CyberCIEGE, the thesis will pursue how its software application can be used to visualize networks and traffic flows. The input for this will be derived from intrusion prevention and detection systems and network protocol analyzers such as Snort or Wireshark. Conceivably, the added feature of being able to visually "see" the networks one is trying to analyze and defend, and the flow of data within these networks would be extremely beneficial to users. It is hoped that the users of these software packages can gain increased “situational awareness” and an overall better appreciation for the networks they are studying, analyzing, and ultimately, trying to defend from attack.

c. Visualization Tools Evaluation

A further goal is to evaluate the capabilities of the Xj3D SAI for programmatically creating three-dimensional visualizations. Currently, there are only a small number of projects created thus far that implement X3D via Xj3D purely programmatically, and without loading and manipulating individual X3D files. This method of X3D application is quite different from creating, loading, and manipulating a stand-alone X3D file. Therefore, this thesis will seek to explore how well Xj3D performs in this capacity, and will provide a framework for how the technology can be implemented.

2. Proposed Solutions

To address the goals in the previous section, the authors will design and develop a Network Topology and Attack Visualizer (Three Dimensional), which will be referred to as NTAV3D throughout this document. The main objective of the NTAV3D application is to be a network visualization tool that allows network administrators, and those interested in learning about securing networks a way to conceptualize this complex system of computing. NTAV3D will provide a three-dimensional view that correlates to the "office view" in CyberCIEGE, but that focuses on network interconnections between components. The tool will then display the actual flaws and malicious software that may exist on these components, while also conveying which asset or assets were attacked and how they were attacked. A framework will then be developed for how NTAV3D could interact with some of the network tools mentioned above to provide similar visualizations.

D. LIMITATIONS

Visualizing the actual origins of cyber attacks is a related, but separate problem that is outside the scope of this thesis, and therefore is not included in NTAV3D's capabilities or the discussion therein.

Additionally, utilizing NTAV3D to provide network visualization to IDS or protocol analyzers is comparatively more difficult than providing the visualization to CyberCIEGE. Chapter V will further address these specific limitations.

Finally, NTAV3D is not meant to perform as a network topology and management, or optimizer, tool. Other tools already exist in these areas that allow network administrators to manage and optimize their networks. Products, such as Hewlett-Packard's "HP OpenView," and SolarWinds suite of network management tools are examples of applications that meet this need. More information on these tools can be found at http://h20229.www2.hp.com and http://www.solarwinds.net respectively (both accessed May 2007).

E. THESIS ORGANIZATION

Chapter II reviews background information in network visualization, including how networks and network attacks have been visualized in the past. This information is reflected in how the NTAV3D visualizations are implemented. Chapter III provides background on the application technologies that were used to develop NTAV3D. Additionally, it explains the decisions that were made with regard to the use of each technology. Chapter IV explains the decisions made for the thesis visualization application, based partly on feedback, and the research of Chapter II. Chapter V discusses how the technologies previously introduced were actually used to implement the visualization, as well as evaluations of these technologies. Finally, Chapter VI provides recommendations for future work as well as suggests ways the NTAV3D application can be expanded and improved. The chapter also includes the results of this thesis, and evaluations on the success of the implementation detailed in this document.

II. NETWORK VISUALIZATION BACKGROUND

A. WHAT IS VISUALIZATION

Visualization can be defined as the cognitive process, performed by human beings, which brings meaning to data (Spence, 2001; Keller, 1993). When performed by a human, visualization is achieved by creating an ephemeral mental model. Mental models created by humans are cannot be easily shared or reproduced, whereas a visualization created via a computer is reproducible and easily distributed. It is important that a visualization be reproducible and distributable so that a large number of people can benefit from it. With this in mind, the context of the term visualization throughout the remainder of this thesis will refer to a visualization constructed via computer. Scientific visualization is a related field where physical objects or phenomena are represented, usually in simulated 3D (Spence, 2001). The goal of any visualization is to improve the user's understanding of the subject. The basic process of visualization is depicted in the figure below.

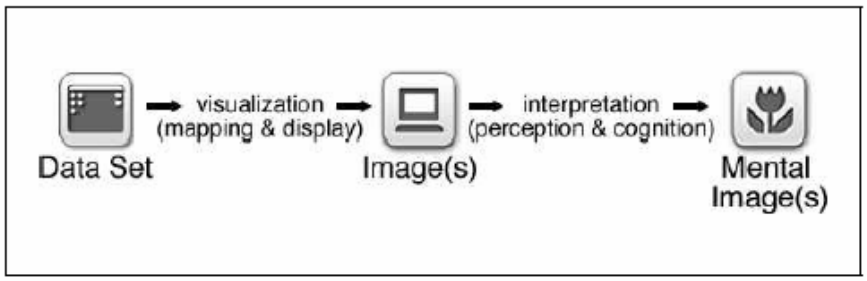


Figure 1. The visualization process (From Holmberg et al, 2006).

The figure above illustrates the basic visualization process beginning with the data set, which is converted to a visual representation to enhance the end user's understanding of the data set. There are two elements that comprise the basic framework of every visualization. These elements are the datasets and the visual representation method.

1. Datasets

The origin of a data set can come just about anywhere, but the most common origins are from laboratory or simulation data, or output from sensors out in the field.

The data is usually in the form of a series of numbers. Although data in the form of numbers is useful when performing a look up or a comparison, forming an ephemeral model from them is difficult. The difficulty can often time be attributed to the sheer size of the data. Employing a computer visualization helps to overcome the size difficulty and facilitate the ephemeral model building process.

2. Representation Methods

There are numerous methods for visually presenting numbers with pie charts, bar graphs, scatter plots, and histograms being some of the most common. Every method has strengths and weaknesses. There is no perfect presentation method that will adequately visualize all the different types of data that exist. The correct representation method is dependent upon the data.

a. Parallel Coordinate, Mosaic, and HyperBox Plots

One method for representing datasets is a parallel coordinate plot where each variable is given its own axis. Each line represents a single data entry. This representation method can handle quantitative data as well as categorical data. Relationships can be obtained by observing the appearance of the plot. The example in Figure 2 has six variables and six data entries. Observing this example plot, it appears that a correlation exists between variables B and C because none of the lines between them cross and all the lines have very little slope. Looking at variables A and B, it appears that an inverse correlation exists because the lower the point on axis A resulted in a larger value on axis B, and vice versa. The ordering of the axis directly influences how easily the same inferences can be made. Following the "a" data entry from one end to the other can be difficult based on the number of data entries and the number of lines crossing. Values can be obtained from the plot if the axes are marked with upper and lower bounds. These values would be placed at the top and bottom of each axis.

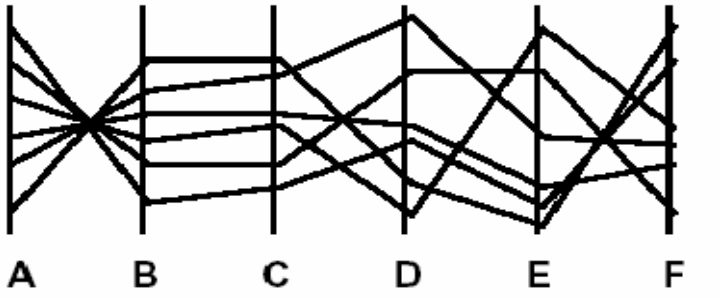


Figure 2. Parallel coordinate plot with six variables (From Spence 2001)

Another representation method is a mosaic plot, which uses rectangles with proportional heights and widths based on the data. This technique is useful when dealing with high-dimensional data. The example plot in Figure 3 shows data concerning the Titanic disaster.

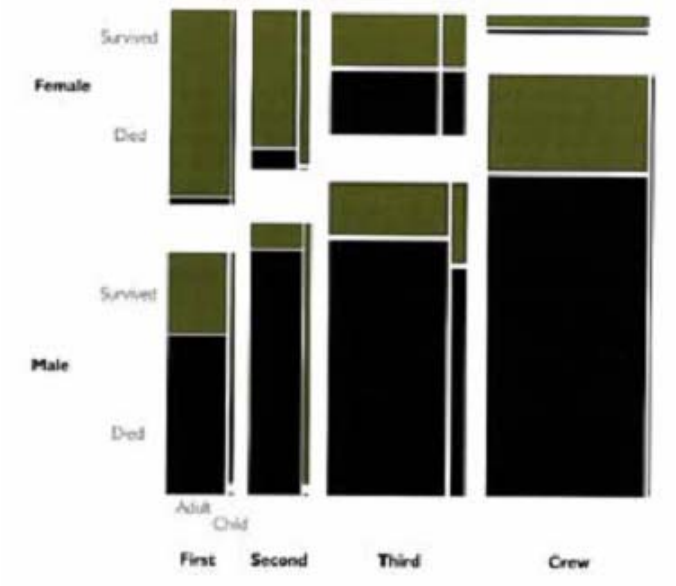


Figure 3. Mosaic plot of Titanic data broken down by gender, age, class and survival (From Spence 2001)

The green represents people who survived and the black represents the people who died. The top portion of data is females and the bottom is males. The four columns represent first class passengers, second class passengers, third class passengers, and the crew, respectively. The small second column represents children, next to each of the larger columns representing adults. A striking observation is the percentage of

women who survived as compared to men. Many other observations can be gleaned from this figure by simply discerning patterns or anomalies in the figure. Unless values are placed within the rectangles, it is difficult to extract numbers from these plots.

Both of these methods can be static displays, as depicted in the previous two figures, or interactive displays. The interactive versions create new static displays based off user interaction. Many people believe that dynamic or interactive graphics are invaluable tools, which aid the process of understanding and finding patterns or anomalies (Becker, 1990). The reason is that interactive graphics allow the user to see the figure from different angles or different configurations. Different views may reveal relationships that were not visible in other views. One of these hidden relationships could potentially provide the information needed to increase the user's understanding. Additionally, "the Theory of Multiple Intelligences (Gardner, 1985) implies that teaching with visual and other components can make learning more effective" (Baxley et al, 2006).

A hyperbox (Alpern and Carter, 1991) is constructed such that all possible pairs of variables are shown plotted against each other (Spence, 2001). The result resembles a solid three-dimensional object with many rectangular faces. Each rectangular face is a separate pairing of two variables. Because all possible combinations are shown, the ordering problem that arises with the parallel coordinate plots is avoided. This method is useful for multivariate data sets.

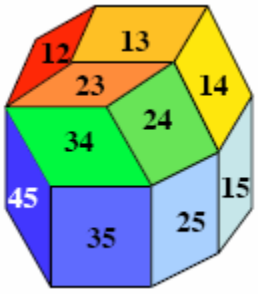


Figure 4. A five dimensional hyperbox (From Spence, 2001)

b. Node and Link Diagrams

The method of interest for this thesis is a node and link diagram. The nodes represent entities of interest and the links represent the relationships between entities (Hansen and Johnson, 2004). These relationships include raw physical

measurements, computed aggregates, or abstract quantities (Hansen and Johnson, 2004). The nodes can be portrayed as boxes, circles, points, or any other glyph—a graphical object—which makes sense for the given visualization (Eick, 1996). Links are normally portrayed as lines connecting one node to another.

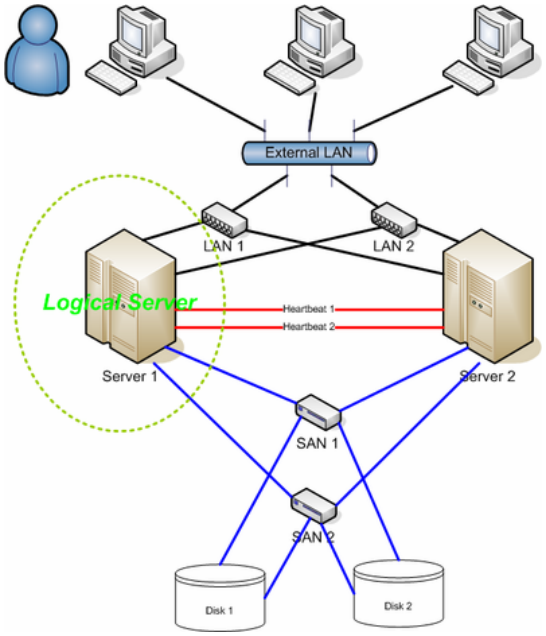


Figure 5. Example Node link diagram obtained from http://www.answers.com/topic/high-availability-cluster on May 21, 2007.

The example diagram in Figure 5 is a simple representation of a high-availability cluster. As seen in the diagram there are a handful of computers, routers, and servers interconnected with colored lines. Some of the visualization techniques employed in node link diagrams were adopted, but in order to correctly represent the physical layout of the network this representation had to be extrapolated into a three-dimensional environment. As opposed to connecting components with straight lines as shown in Figure 5, it was necessary to devise an alternative connection scheme in order to minimize ambiguity and display clutter in the three-dimensional environment. The solution and its benefit are discussed in detail in Chapter IV.

B. CURRENT NETWORK VISUALIZATION RESEARCH

Because networks are becoming increasingly large and complex, much of the research efforts invested in network visualization these days are focused on solving the

problems associated with visualizing large networks. These problems include scalability and alleviating display clutter, generating the appropriate topology, and achieving real-time visualization results. The definition of what constitutes a large network may vary slightly from source to source, but in the context of this thesis, networks with more than 100 nodes will be considered large networks. Large networks are not an issue within CyberCIEGE because none of the networks created by game players approach the 100 node mark. Large networks are likely to be an issue when dealing with output from intrusion detection systems in the real world.

1. Solutions

a. Reducing Display Clutter

With a finite amount of display space on a computer screen, visualizing the entirety of a large network in manner clearly discernable to the end user is a difficult, if not impossible, task to accomplish due to display clutter. The American Heritage College Dictionary defines clutter as "a confused or disoriented state, caused by filling or covering with objects." This definition simply means that clutter beings on confusion due to the number and spacing of objects within a scene. Scaling the network, controlling what is displayed, color coding, and utilizing three-dimensional space are four commonly employed methods to reduce display clutter.

The first common method to reduce the amount of display clutter is to scale the large network into a smaller, more manageable size. This method results in a smaller, less detailed network that can be displayed in a more clear manner.

The second common solution applied to this problem is to display only the network components that are visible within the current view. This method can be thought of as a level-of-detail method where a detailed representation of a particular area is shown only when focused upon by the user. The user controls what level of detail is shown by navigation through the scene.

The third method is to use a color coding scheme to represent numerical values or other information which may occupy large amounts of screen real estate and may also obscure other more important details is another method employed to reduce ambiguity in the scene (Kershenbaum and Murray, 2005). By replacing physical

numbers with a color coding scheme, the amount of information displayed is reduced, yet the amount of information that can be gleaned from the display is not reduced. In NTAV3D, links are color coded to visually differentiate between separate networks instead statically listing the name of the network next to the line.

The fourth reduction method is to use a three-dimensional representation as opposed to a two-dimensional display. Visualizing in three-dimensional space does introduce some different issues. Three-dimensional displays are often times confusing, difficult to navigate, and cause the user to loose a sense of overall context (Cox et al, 1996). One method to mitigate navigational difficulties while maintaining context is to restrict the navigation capabilities and make the display as user friendly as possible (Cox et al, 1996). Restricting navigation such that the user cannot travel inside a piece of geometry and maintaining the camera's focus on the scene as the user navigates through the scene will keep the user from becoming disoriented in the three-dimensional world. In NTAV3D, the decision as to how to allow the user to navigate through the scene was additionally constrained by maintaining a close relationship to controls already in place within CyberCIEGE. Similar control schemes allow the user to transition from one application to the other without having to memorize different control patterns.

b. Generating Correct Topology

Although reducing display clutter increases the likelihood of information being extracted from a visualization, the chosen topological display of the data directly impacts the relationships the user is able to take in.

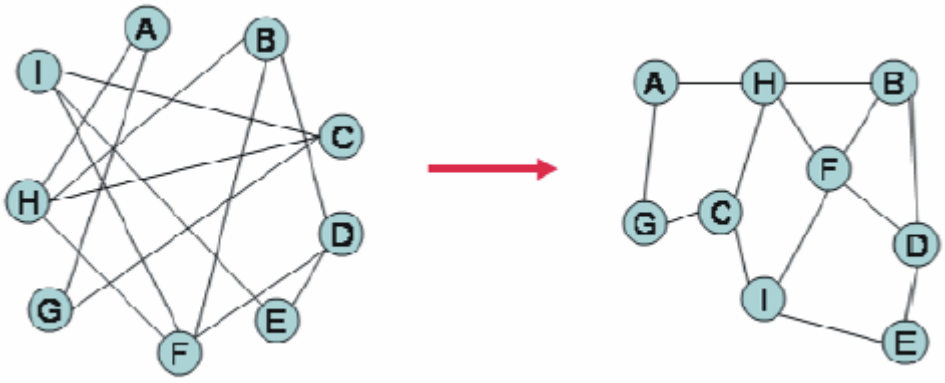


Figure 6. Two topologies for the same nodes (From Kershenbaum and Murray, 2005).

The model on the right in Figure 6 shows the same relationships was the model on the left, but positions the nodes such that no lines cross.

The same nodes displayed in a slightly different topology can convey a different meaning. Topologies can be hand generated, but in most cases, they are generated by an algorithm to meet certain criteria. Many algorithms are designed to meet one or more of the following criteria:

- Minimize the number of links crossing
- Maximize the depiction of symmetry
- Minimize the number of link bends
- Maximize the minimum angle between links leaving a node
- Maximize link orthogonality

The level of importance for each criterion is based on what is being visualized. In cases where physical location is important, the criteria above become unimportant because the positions of nodes cannot be changed to meet the aforementioned criteria. Geographic data is a good example of when physical location is important. Because the physical location of nodes is specified by CyberCIEGE, straight lines could not simply be drawn between nodes because eliminating crossing lines was also import. This led to the development of a routing algorithm.

A group of researchers took different approach to placing nodes. Instead of using criteria that create an aesthetically pleasing display, such as minimizing the number of crossing links, these researchers looked into developing an algorithm that takes into account the weight (strength) of the links (Eick and Wills, 1993). With this approach, the strength of the link between two nodes is what determines their relative position from one another. They determined that the algorithm would have an inverse relationship such that nodes connected by stronger links would be closer together than

nodes connected by weaker links. The desired inverse relationship is achieved by directly minimizing the function

$$\Sigma (w_{ij} - 1/d_{ij})^2 = \Sigma (d_{ij}w_{ij} - 1)^2 / d_{ij}^2$$

where w_{ij} is the weight of the link and d_{ij} is the displayed length of the link (Eick and Wills, 1993).

The developed algorithm also takes advantage of hierarchical relationships by placing the root node then placing all the sub-networks generated by its children.

A variety of other tools exist that focus solely on the generation of network topologies. BRITE is a free topology generation tool from Boston University that is implemented in both Java and C++, but is no longer supported by its developers. To download or for more information about this tool please refer to this website: http://www.cs.bu.edu/brite/. GT-ITM is another topology generation tool that was developed at Georgia Tech and is freely available for download (http://www.cc.gatech.edu/projects/gtitm/). Inet is a more recent tool developed at the University of Michigan for generating Internet topologies. The latest version (3.0) was released in 2002 and can be downloaded at no charge from http://topology.eecs.umich.edu/inet/. Other tools and algorithms exist for generating topologies, but these are three popular tools that are available to anyone.

c. Real-Time Visualization and Data Reduction

The size of datasets has grown tremendously over the past ten years with increased processor speeds and hard-drive size. And despite these technological advancements, real-time visualization remains a difficult problem. In some cases, this problem can be overcome by data streaming which is characterized by processing independent subsets of the larger dataset. Streaming data is accomplished by breaking large files into smaller files and loading these files in series. The small files can be loaded in real time, while the single large file cannot.

Data reduction filters out data that is not relevant to visualization in order to make the size of the dataset manageable. Three traditional methods used to reduce the amount of network data to a manageable size are:

- Aggregation which is used when large numbers of links or nodes are present
- Averaging when dealing with large numbers of time periods
- Threshold and exception reporting used to detecting changes (Becker et al, 1995)

When using a data-reduction method, the possibility exists that important information may be obscured or lost during the reduction process. Instead of reducing the size of data sets and possibly loosing valuable information, research is being done to develop more efficient algorithms that reduce the computation time of popular force-directed layout algorithms used on large networks (Au et al, 2004).

2. Available Network Visualization Tools

Network visualization tools are continuously being produced and tweaked in order to adapt to changing needs and to keep pace with the advances and changes made in the network field. In the mid 90s, people at AT&T Bell Laboratories came up with *SeeNet*, which consists of three graphical tools for visualizing network data (Becker et al, 1995). The example network used throughout the paper involves 110 nodes and over 12,000 links. Instead of trying to create a single view that depicts all the necessary information at once, they chose to present the information in five separate views (Becket et al, 1995). One view is a network map showing the entire network, a time slider view for navigating through the time varying data, an interactive color scale view for manipulating link colors, a bird's-eye view for a more detailed view of areas on interest, and a control panel view for other types of analysis and user controls. These views were all two-dimensional. A year later, they added a three-dimensional view for better geographic context (Cox et al, 1996). SeeNet is an application that was described in the previous section and is used for visualizing network data for communication networks.

Another tool for network visualization is called VLNT (visualizing large network topologies). This tool is designed to assist network managers in analyzing and improving Internet routing topologies. Incorporated into this tool is a novel hybrid layout algorithm for visualizing large network topologies (Au et al, 2004). The algorithm uses the inverted

self-organizing map (ISOM) for initial layout, and then uses the Kamada-Kawai algorithm to refine and fine tune the layout. ISOM is stochastically based competitive learning algorithm that is highly versatile (can produce a 2D or 3D layout), consumes relatively few computational resources, and is not application specific (Meyer, 1998). The Kamada-Kawai algorithm is a force-directed layout algorithm that models the network as a set of springs and tries to find a layout that minimizes the energy in the springs (Au et al, 2004). By combining these two previous approaches to node placement and introducing a new termination criteria (edge-tension gradient) this new approach leverages overall layout structure of the ISOM method with the unclustering ability of the Kamada-Kawai method. The new termination criterion is "based on the ratio of the average edge length in the layout to the ideal edge length" (Au et al, 2004). The ideal edge length is calculated based on the dimensions of the display in order to keep the from being too large for the display. Termination is reached once this ratio falls below a specified threshold. The recommended thresholds were determined through empirical testing.

The network animator (Nam) is a visualization tool that utilizes animated packet flow for taking in large amounts of information quickly and visually identifying patterns in communication to promote understanding of causality and interaction inside networks (Estrin et al, 2000). Nam has three different methods for generating the network topology. The first and most common method is an automatic layout algorithm based on a spring embedded model. The second method, used only for small topologies) is relative and allows the user to specify the relative directions of links (left, right, up down) and positions the nodes accordingly. The third method is wireless and associates a node with its physical location, but this method typically lacks explicit links. The animated packets appear as rectangles with arrows at the front to indicate direction of travel and the path is determined by trace events that indicate when a packet enters and leaves links and queues.

C. CURRENT NETWORK ATTACK RESEARCH

As society continues to move forward in the Information Age, computers are being integrated into more facets of everyday life than ever before. Additionally, these

systems are being linked to one another creating new and complex networks. With all these systems and networks, information assurance and system security are critical areas of research. Within the information assurance and system security realms, the main research areas include modeling and understanding multi-stage attacks, developing detection models for novel attacks while having a low false alarm rate, real-time detection and analysis on high speed network traffic, and tracing the origins of attacks.

1. Solutions

a. Modeling and Understanding Multi-Stage Attacks

A number of research efforts are focusing on classifying and categorizing the methods used to exploit vulnerabilities in systems during multi-stage network attacks (Tidwell et al, 2001). Multi-stage attacks are a type of blended attack with the ability to infiltrate protected systems by eluding threat detection systems (Dawkins and Hale, 2004). Tidwell and his fellow researchers used parametric attack trees in conjunction with a system specification language to support vulnerability assessments and attack visualization that can be extended to provide real-time attack notification and monitoring services (Tidwell et al, 2001).

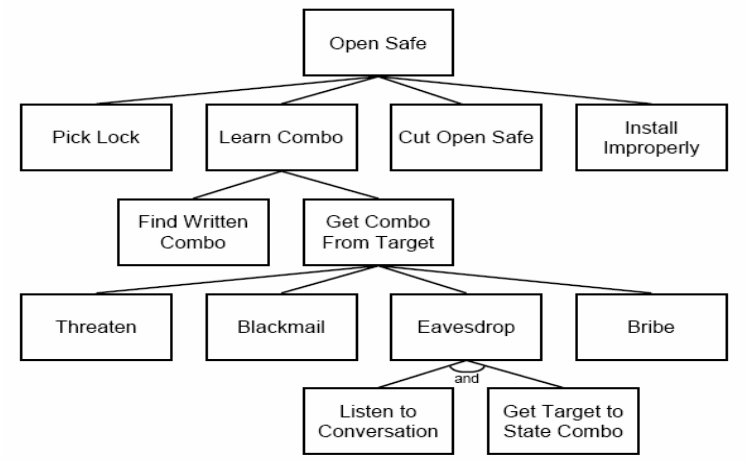


Figure 7. Example attack tree from Bruce Schneier at Counterpane Systems.

Attack trees represent attacks and countermeasures as a tree structure, where the root node is the goal of the attack and the leaf nodes are the attack methods (Schneier, 1999). Figure 7 shows a very basic example of the structure of an attack tree. More information

can be added to each node such as a weight reflecting the probability of success or with the cost associated with the node (Tidwell et at, 2001).

Dawkins and Hale supplemented a network model with a capabilities model to express the resources and skills of the attacker. The vulnerability model uses the two previous models to determine vulnerabilities, exposures, and exploits. An iterative attack chaining process identifies a series of possible attack scenarios, which are used to construct attack trees from the perspective of the attacker. Probability values are assigned to the given vulnerabilities during the analysis phase. These values help to identify the most probable avenues of attack for multi-stage attacks.

b. Detection Models for Novel Attack Schemes

IDSs are continually playing the catch-up game to detect the continuous and ever changing threats posed by computer attackers. Attackers are continually developing new methods to bring down or penetrate networks. New detection schemes are being developed and tested in order to increase the probability of detecting novel attacks while lowering the false alarm rate. Of course, the ideal scheme would have a 100% detection rate with a 0% false alarm rate. Such a model does not exist and realistically is near impossible to achieve. Anomaly detection and pattern or signature recognition are the two common detection schemes implemented for intrusion detection. Pattern recognition is efficient and accurate, but only useful in identifying known attack patterns, while anomaly detection methods can be used to identify novel attacks. One recent method utilizes Principal Component Analysis, a well-established technique for reducing dimensionality and multivariate analysis, to reduce the data attributes down to two or three dimensions (Labib and Vemuri, 2004). Analysis techniques are subsequently applied to the linear combinations of Principal Components to determine when attacks have occurred. Another method expands upon existing intrusion detection work by generalizing the activity properties of data into a frequency property, a duration property, and an ordering property (Ye et al, 2001). The method focuses statistical methods including decision trees, Markov chains, and Hotelling's T^2 test on analyzing these focus areas. The study found that the frequency property is essential for identifying

attacks, but when coupled with information from the ordering property, its identification power is increased (Ye et al, 2001).

In a later study, researchers investigated the effects of building a long-term profile of normal activities on a machine in order to compare recent and current activities with the long-term profile to try to detect significant deviations (Ye et al, 2002). Detection is based on Hotelling's T^2 test, which is able to detect both counter-relationship anomalies and mean-shift anomalies.

Research has been done to visually combine information obtained from an intrusion detection monitoring environment with network traffic information. The information obtained can be used to identify bottlenecks, failures, and wasted resources on the network. The research conducted by Erbacher is geared towards simple two-dimensional visualization techniques for continuous online monitoring of the network by system administrators and network managers. The goal is to aid network managers' "ability to asses the effectiveness of the network infrastructure and plan long range infrastructure management as well as deal with short term and immediate crisis, such as intrusions and misuses" (Erbacher, 2002).

c. Real-Time Detection and Analysis

Having good detection models is one thing, but being able to detect attacks while they are occurring and providing users with helpful information is another matter. In a recent study, ongoing attacks were visualized by mapping source IP addresses, destination IP addresses, and destination port numbers to a three-dimensional Cartesian space (Kim et al, 2004). These values are easily obtained from most Internet packets. From this plot, denial-of-service attacks, host scans, and port scans are easily seen. One short coming that was identified with this presentation method is that low intensity attacks and attacks occurring near dominant legitimate traffic are hard to identify from the three-dimensional plot. To over come this problem they implemented an original "pivoted movement" algorithm which tracks the three values of these packets and watches for instances where two of the coordinates remain fixed, while the third pivots around (Kim et al, 2004).

d. Tracing the Origins of Attacks

The last main area of focus in network attacks is tracing the attack to its source. Often, attackers will forge their source IP address in packets in order to hide their identity. Proactive tracing is done before an attack is detected, while reactive tracing occurs after an attack is detected. Two proactive tracing methods are packet marking and messaging. Reactive tracing methods include hop-by-hop tracing, hop-by-hop tracing with an overlay network, IPsec authentication, and traffic pattern matching. A recently developed approach augments hop-by-hop tracing with "datalink-level identifiers such as Ethernet's media access control (MAC) address, ATM's virtual path identifier/virtual channel identifier (VPI/VCI), and frame relay's datalink connection identifier (DLCI) to identify nodes in the packet's path" (Baba and Matsuda, 2002). Because it is difficult for an attacker to forge the datalink-level identifiers of intermediate forwarding nodes, it is easier to trace packets back to their source using datalink-level identifiers. Because of the complexity of this issue and the fact that this information is not available within CyberCIEGE and not provided by intrusion detection systems, visualizing the origins of attacks is not presented in NTAV3D.

2. Available Network Attack Visualization Tools

A wide variety of intrusion detection systems are available for download on the web. Some are free like Snort, AirSnare, Prevx Home, WinPatrol, and System Safety Monitor, while others cost money like Process Guard, RegRun Gold, and Geek Superhero.

NIVA is a network intrusion visualization application designed for interactive investigation and detection of structured attacks across time and space that utilizes three-dimensional displays (Nyarko et al, 2002). The application takes in output from an intrusion detector in simple ASCII or database format. This detection list is parsed according to the selected model. A node placement list is generated assigning locations for each node in three-dimensional space by employing either the IP-Space algorithm, the "spring technique," or the "helix technique." The IP-Space algorithm places components based on relationships gathered from the IP addresses of components. The "spring" technique assigns location based on the strength of a nodes connection to a central node,

while the "helix" technique assigns locations in helical pattern. Afterwards, a node-link map is created. The scene is rendered in OpenGL and the user then has the ability to manipulate both the displays and the data through the user interface.

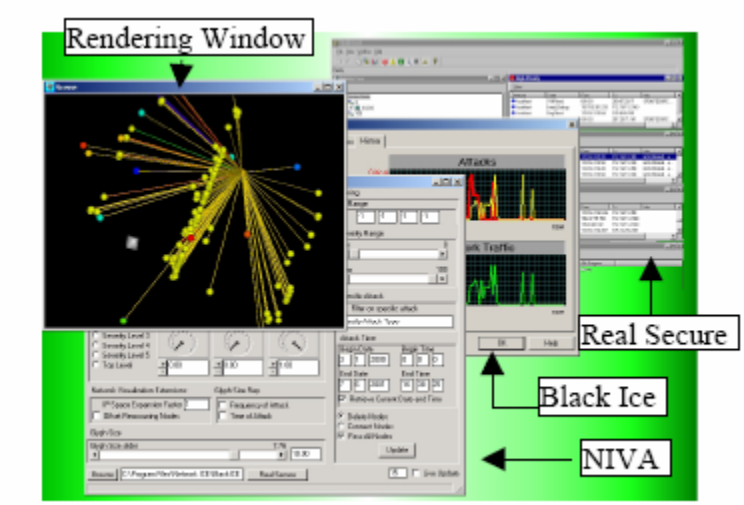


Figure 8. Typical NIVA Session (From Nyarko et al, 2002)

The experiment specification and visualization toolkit (ESVT) was developed at Penn State University for use by experimenters conducting interactive experiments on network test beds (Li et al, 2006). It is comprised of a topology builder, a TCL script generator, and various visualization tools. Users have the choice of manually coding the topology themselves, or they can import topologies generated by Inet, GT-ITM, BRITE, or tiers topology generators (Li et al, 2006). ESVT is able to visualize traffic dump data and node status logs with the main method of visualization consisting of a time-series animation with adjustable time steps. A more in-depth visualization is available for viewing the data at a more refined level such as the number of UDP packets during a specific time period. This is achieved by selecting a link and defining the filter rules.

LAN attacker is an application developed primarily through North Carolina A & T University, but also with help from Wesleyan College, to visualize the three major attacks on Local Area Networks: ARP Poisoning, Switch Port Stealing, and Mac Flooding (Baxley et al, 2006). The visualization provided includes a two-dimensional model of a simple local area network (about ten nodes) and simulates network traffic by

having virtual packets, containing the Mac address and IP address of both the sender and Target, that travel through the network model along the a virtual path represented by bold black lines (Baxley et al, 2006).

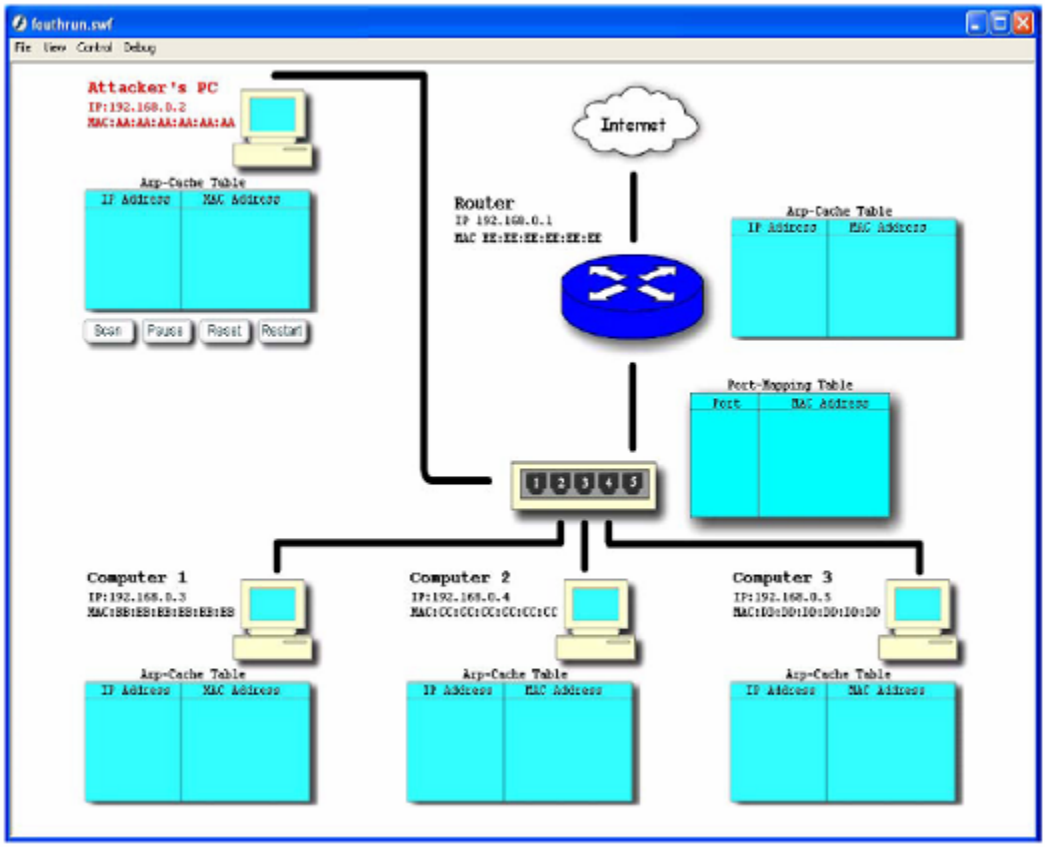


Figure 9. LAN Attacker components (From Baxley et al, 2006).

The figure above shows basic interface and network topology of the LAN Attacker application. During the attack phase, users can select which method of attack will occur. Each of the attack methods is accurately modeled for this simple configuration. Users are able to interact with the application by pausing the scenario at any point. This allows the user to examine the animated packets' details, update ARP Cache Tables, and update the Port Mapping Tables. Additionally, text boxes appear as users' mouse over the various devices in the scene. The application also features a short quiz upon completion of the scenario to test the comprehension of the user.

D. KEY TAKE AWAYS FROM CURRENT RESEARCH

In order to create a successful visualization, one must display the information in a manner that allows users to easily understand the relationships and other statistics being displayed with as little confusion as possible. One useful method to reduce confusion levels is to only display the necessary information, and if possible, keep away from statically displaying numerical or textual information as they may take up valuable screen real estate. Color coding is one possible alternative to displaying numerical values when only relative estimations of values are necessary.

Aside from the actual display itself, incorporating user interaction where possible is important because interaction between the user and the display increases the opportunities for the user to find a way to understand the information being displayed. This includes viewing the scene from multiple angles so that information occluded or partially obscured by one particular view is seen in a different view. Animation is a useful tool when trying to visualize network attacks.

III. NTAV3D TECHNOLOGY BACKGROUND AND DECISIONS

A. INTRODUCTION

Since the product of the research of Chapter II is a working software application, it is therefore prudent to explain how the software was built, and why certain development choices were made. The intent is not to be a primer on the technologies discussed below, but to explain how they relate to NTAV3D, and the reasoning behind why they were selected. The motivation of the following is to provide the reader a basic understanding of the underlying technologies that make up NTAV3D. References are included for further examination of each technology.

B. XJ3D

1. Description

As introduced in Chapter I, Xj3D is the Java-based API for programmatically implementing, and stand-alone browser for displaying, the X3D graphics format, and is developed by the Web3D consortium. (The Xj3D Project, April 2006) As such, NTAV3D utilizes both components of Xj3D. The browser component is currently implemented in a stand-alone application, which can be launched independently. However, the browser, and NTAV3D specifically, has the capability to be embedded within a program as well. Additionally, the API component, again known as the SAI or Scene Access Interface, is what is used as libraries in NTAV3D's Java code to create and manipulate an X3D scene graph in a purely programmatic way.

Xj3D is in a continual state of development. The most stable release is version 1.0 dated 15 April 2006, but "snapshot" releases, which contain new features and improvements, are continually added. For this, and reasons stated below, NTAV3D utilized the "bleeding edge" of the codebase. Thus, the latest version of Xj3D implemented within NTAV3D is the "development snapshot" from 12 April 2007. More information on Xj3D can be obtained from the project's website at http://www.xj3d.org (accessed May 2007).

2. Decisions

When choosing which software architecture to use to create a three dimensional scene, the authors were looking for a package that did not have a steep learning curve, and one that would be easy to integrate with CyberCIEGE first, and then network tools second. Due to the authors' background in X3D via previous coursework, and the fact that both Xj3D and CyberCIEGE are implemented in Java, Xj3D became an attractive option.

One example of an alternative to Xj3D was OpenSceneGraph, which is another three dimensional graphics package that is implemented with C++ and OpenGL. However, OpenSceneGraph has a fairly steep learning curve, and with little previous experience, the authors would have been required to learn the intricacies of programming with OpenGL, and C++, two programming constructs the authors were not as familiar with. Xj3D actually also implements an OpenGL renderer as the method of displaying the three dimensional scene. But, one of the benefits of the Xj3D libraries and implementation is that it handles all of the OpenGL setup "behind the scenes," without requiring much work from the developers other than setting parameters for the browser component. Additionally, the authors were already familiar with the X3D system, and felt it would be a fairly easy transition to implementing X3D programmatically via the Xj3D SAI. For further discussion of this experience, see Chapter VI, where an evaluation of Xj3D is offered.

C. JAVA

1. Description

The main technology utilized to create NTAV3D was Java. Java is an open source, multi-platform programming language. Being open source means that Java's program source code is freely available to be extended or enhanced without having to pay large licensing fees. This also means that programs created with Java have fewer restrictions on their ability to be redistributed.

The application was compiled with the Java SE JDK (Java Development Kit) version six update one (otherwise known as Java 1.6). However, the application is

compliant with and will run on any Windows computer running Java version 1.5 or later. For further information about Java, and to obtain thee Java runtime needed to execute the application, one can visit the Sun Java website at http://java.sun.com/javase (accessed May 2007).

Additionally, the primary IDE, or integrated development environment, chosen was Eclipse, which is available from http://www.eclipse.org (accessed May 2007). However, since the application is written purely in Java, any development environment could be used, and in fact, some application testing and debugging was conducted using the NetBeans IDE, available from http://www.netbeans.org (accessed May 2007).

2. Decisions

Java was the language of choice for this thesis application, instead of, for example, C++, both because of the authors' comfort level with the language, and due to the previously mentioned fact that the graphical API of choice, Xj3D, is implemented solely in Java. This choice of languages and APIs has many benefits. First, both Java and Xj3D are open source and, as noted above, freely redistributable. This makes them quite viable for a Department of Defense program or for further expansion of capabilities by an interested community of developers later. For example, since NTAV3D is Java-based, it could conceivably be made as an applet, and distributed over the web, thus making network visualization quite portable (see Chapter V for more future work suggestions). Additionally, since the application is written in Java and with the Java-based libraries of Xj3D, it can run on almost any computer platform, be it Microsoft Windows, Apple OS X, or different variations of Linux. This is an exciting feature, as it can be used on any of these machines to visualize network activity.

In terms of development environments, primarily, Eclipse was chosen due to the comfort of the authors, and because it contains an excellent revision control system, specifically CVS (Concurrent Versioning System). CVS is a system that allows programmers to work on application code simultaneously by storing the code in a "repository" on a central server. Authors "checkout" and "checkin" code as it is written, which allows everyone to be working off of the latest version, and not one that may have

become obsolete due to a colleagues editing. CVS is recommended for anyone who may be working on programming code with multiple authors as it facilities easier collaboration.

D. XML

1. Description

XML stands for Extensible Markup Language, and is the main way NTAV3D receives data from outside programs, be it CyberCIEGE, or other network analysis tools. XML is a human-readable, non-proprietary file format that is most often used to exchange data between different computer programs or systems. XML is based on the concept of a tree based, hierarchical file structure, where a tree of descriptive "tags" surround data. Additionally, XML schemas can be developed. These schemas are how XML becomes "extensible" in that the basic XML construct can be extended to create what amounts to a virtual language all its own, as different schemas can be developed to fit different needs.

In the case of NTAV3D, a document type definition, or DTD, schema was developed to standardize how information from CyberCIEGE could be output as an XML file. It would be this DTD, and the CyberCIEGE XML-based output file that NTAV3D would load, parse, and use as input data to create the network visualization. Because this process is similar for all XML files, this is also how the program could be expanded to enhance intrusion detection systems (see Chapter V for more information on how NTAV3D XML processing works).

To obtain more information about XML in general, one can visit the W3C web-standards body's site at http://www.w3.org/XML (accessed May 2007).

2. Decisions

The decision to use XML as a kind of "middleware" for NTAV3D was made dude to the power of XML. Like Java, XML is platform independent, since it is nothing more than a text file. Additionally, XML is not a proprietary file format, so it is unencumbered by licensing restrictions and other restrictions on redistribution. XML,

due to its tree structure, is also well suited for holding structured data, such as network information, and as such is also fairly easy to parse. Also, through what are known as XSLT, or Extensible Stylesheet Language Transformations, other XML files can be converted to match the specifications of the NTAV3D XML parser, making it easier to import data from other sources besides CyberCIEGE.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. VISUALIZATION DECISIONS

A. WHAT IS BEING VISUALIZED

Before introducing the decisions and reasoning behind *how* elements are to be visualized, a brief discussion of *what* is being visualized is necessary. What are being visualized are attacks on computer resources and the assets stored on those resources. To facilitate this visualization the physical network topology is presented. This topology comes from the CyberCIEGE scenario being played, and with some added work, can be extended to also include topologies described by output from intrusion detection systems.

1. What is a Component Compromise

Component attacks or compromises refer to flaws within the component protection mechanisms, or attacks on the component such as insertion of malicious software, or the physical theft of the component. Malicious software inserted into a component can originate from a variety of sources. In most cases within CyberCIEGE, the origin of an attack is not known. Information about the origin of some malicious software is potentially available within CyberCIEGE, but only in cases where a decision from the user, like not scanning emails for malicious content, results in a system infection. Even when it is the case that the origin is known, the information is not output to the visualization application. As a result, the goal was not to show *how* a component became compromised with malicious software, but that the component *is* compromised. Again, the focus of the application is on visualizing network topology, highlighting flaws or compromised components, and attacks on assets.

The purpose of an intrusion detection systems is primarily to identify how a user's system is compromised with malicious software and/or data designed to exploit known flaws. An intrusion detection system determines that an attack has occurred by observing traffic patterns and anomalous activities, such as port scanning. With most IDSs, the types of attacks that can be identified are limited to known attack patterns. Identifying the specific attack is difficult in some cases. With some additional research and work,

these vulnerabilities and compromises can be depicted using NTAV3D. The main obstacles that currently prevent depiction of these vulnerabilities and compromises are discussed in section B of Chapter VI.

2. What is an Asset Attack?

An asset attack refers to attacks upon assets, which are contained within a component. Component compromises can facilitate asset attacks by providing an avenue by which an attacker can access the asset contained within the component. In some cases, the compromised component is not necessarily the component that contains the asset. For example, a firewall can be breached, leading to disclosure of an asset on some other component. Similarly, compromise of an outward facing web server can lead to the compromise of assets stored on the backend database systems or storage servers. An asset attack is launched by someone (an attacker) with the goal of obtaining the asset or altering it in some way that benefits the attacker. Integrity attacks and Secrecy attacks are the two types of asset attacks that are visualized. Availability attacks (e.g. a denial-of-service) are not represented as asset attacks because these attacks can be deduced by observing component compromises.

a. Secrecy Attacks

A secrecy attack results in the unauthorized disclosure of information. This can occur as the result of a lack of or failure to properly configure protection mechanisms. It can also occur as the result of component compromises described above, i.e., some combination of flaws, malicious software, or physical theft. In all cases, the flow of information is from the component containing the asset to the attacker.

b. Integrity Attacks

An integrity attack results in the unauthorized modification or destruction of information. These attacks occur in a manner similar to secrecy attacks described above. However, the specific modification that the attacker wishes to make to the asset may be encapsulated within malicious software inserted into a component that can access the asset at some later time. Therefore, since the tool does not depict the source of malicious software, the flow of information form the attacker to the asset is not always

depicted. The player might only see the flow of information from malicious software on some component to the asset. In this case, an attacker will not be displayed because this would illustrate the placement of malicious software, which is a facet of a component compromise that is not visualized.

B. GEOMETRIC REPRESENTATIONS

Because the structure of a network is represented well by a node link diagram, a big decision became what to use as nodes to represent the various network components. The logical solution was to represent the various components with actual models of the various components. This is by no means a new or novel idea. This method of representation allows the user to easily and correctly identify components within the visualization.

1. Computers, Servers, and Routers

All computers were modeled to resemble a typical computer tower. A monitor was not included in the model because tower representation was hypothesized to be sufficient enough for the user to correctly identify the model as a computer in the scene, although no user studies were done to confirm this hypothesis. A screen shot of the actual model developed to represent a computer is displayed in Figure 10 below.

Figure 10. A screen shot of the front of the computer model

Because a server is basically just a computer tower that has different software and in most cases lacks a graphical user interface, the same computer tower model was adapted to also represent servers. In order to reduce the confusion when trying to differentiate between servers and computers, without creating a separate model servers,

the aspect ratio of the server was altered significantly. The dimensions of the computer are 0.6 X 1.2 X 0.8 meters, while the dimensions for the server are 1 X 1 X 1 meters.

Figure 11. A screen shot of the front of the server model

Compared to Figure 10, the Figure 11 model appears tall and slender, while a sever model looks like a cube. In many cases in the real world a number servers are stacked atop one another inside a rack. This is also the case in some CyberCIEGE scenarios. In order to reduce ambiguity and allow CyberCIEGE users to distinguish individual servers from one another when stacked, black lines had to be added to the textures used on servers.

Figure 12. Multiple servers in a stacked configuration

The figure above shows four severs stacked on top of one another. The black lines wrap all the way around the model such so that individual servers can be visually resolved within the server stack. These lines mark the border between servers.

Routers are an essential component with the framework of a network. The routers were modeled after the average router that can be purchased from any retail vendor. This

model is easily identifiable when compared to computers and servers. The figure on the following page shows a side-by-side comparison of these three network components.

Figure 13. A computer (left), router (middle), and server (right)

2. Internet Cloud

Physically modeling the Internet would be a challenge in and of itself. Additionally, a physical model of the Internet might not be correctly identified as such because people have not actually seen a physical representation of the Internet. A widely accepted and widely used method for abstractly representing the Internet is to use a cloud. A simple search on Google Images for "Internet Cloud" returns over 500,000 results. A physical model of a cloud is used to represent the Internet within the visualization. A model of the Internet is necessary because at some point a connection is made to the Internet. Instead of having lines that connect to the Internet meeting at an arbitrary point in space, these lines will meet up within the Internet cloud model that is shown in the figure below.

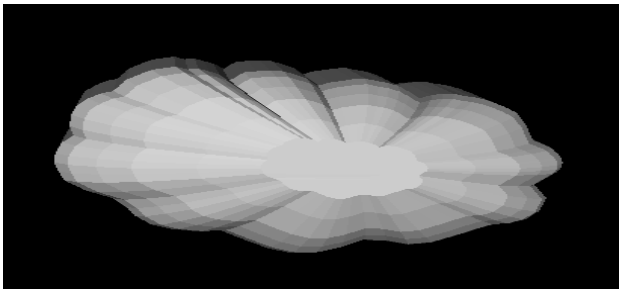


Figure 14. Internet Cloud model

3. Walls

Within the visualization, walls were represented by paper-thin planes. These walls denote the zones in the scenario, which are the abstraction, used to represent physical security in CyberCIEGE, and within the visualization, they provide the user with a visual spatial reference frame. They were given a high level of transparency (.99 out of 1.0) in order for geometry such as network lines and to be visible when looking down on the scene from above. Comparing Figure 15 and Figure 16 below illustrates why such a high level of transparency was needed. The level of transparency in Figure 15 is .99, compared to .85 in Figure 16. Also, the network links are clearly visible through the floor in Figure 15 while in Figure 16, the same lines are barely visible.

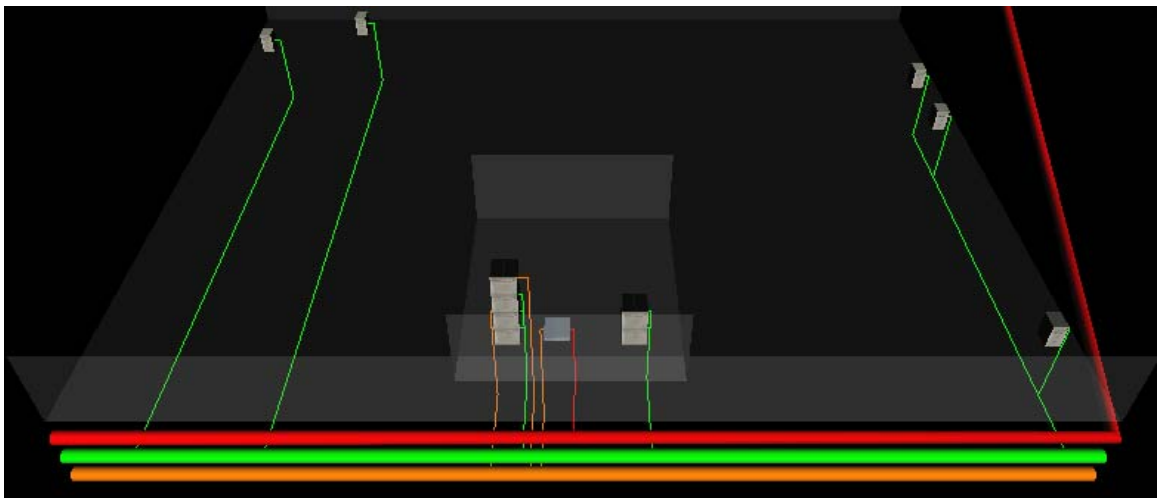


Figure 15. Illustration of semitransparent walls with .99 transparency

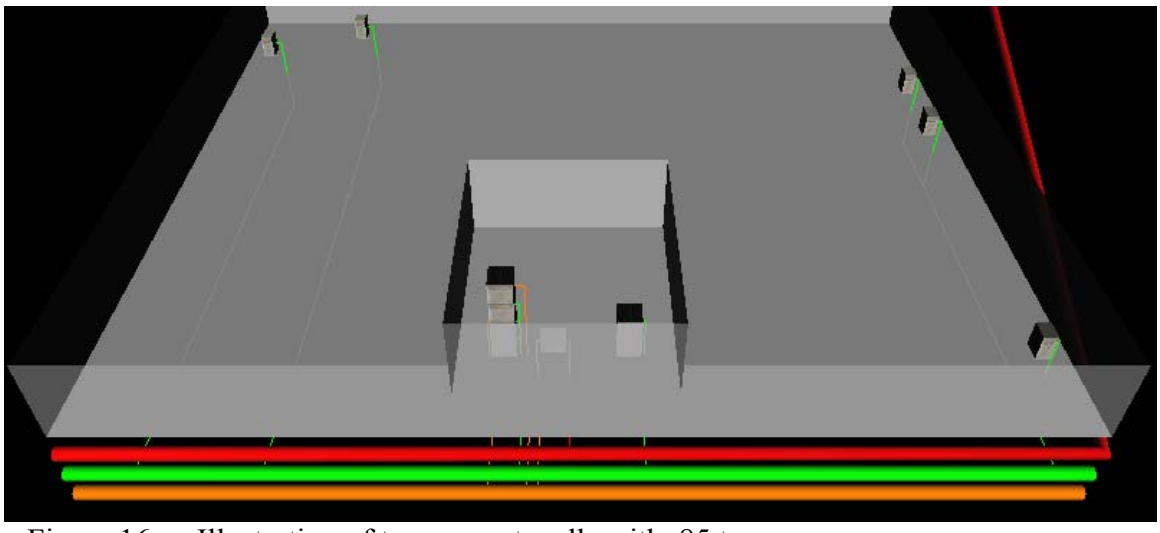


Figure 16. Illustration of transparent walls with .85 transparency

4. Network Links and Main Lines

Figure 15 shows examples of both network main lines and network links. The network links represent connections between components and the network main lines. These connections are denoted by lines that have a width of a single pixel. In order to represent larger traffic flow rates present within network main lines, and to differentiate main lines from links, the main lines were model as large pipes. A radius of 0.3 meters was chosen in order to reduce the sensitivity of the touch-sensor associated with this geometry to a level where the user is able to effectively utilize the mouse-over capability. The difference in size between network main lines and network links is clear when comparing the three network main lines depicted along the bottom edge of Figure 15 with the network links spread throughout the remainder of the scene. Both network links and the main lines were color coded in order to differentiate between different networks present in the scene.

5. Attacks: Component Compromises

Component compromises refer to flaws or attacks that compromise a component itself, while asset attacks refer to attacks upon assets, which are contained within a component. CyberCIEGE simulates a limited number of component compromises, and directly identifies the compromised component and the type of compromise. Within CyberCIEGE, there are five types of component compromises: Trojan horse, virus, operating system flaw, trap door, and physical theft. Models were created to visually represent each of these compromising attacks.

a. Trojan Horse

The Trojan horse was modeled after an image of a Trojan horse. With the focus of this thesis (visualizing network structures and attacks occurring within these networks) in mind, the image to base the Trojan horse model upon was selected based on its simplicity and visual discernability. A simple model is less expensive to render than a complex model with a large number of vertices. The total number of vertices in the Trojan horse model is just under 250. Because a Trojan horse e is constructed from

wood, the model was given a light brown color. The resulting model displayed in Figure 17 is discernable as a Trojan horse and quick to render.

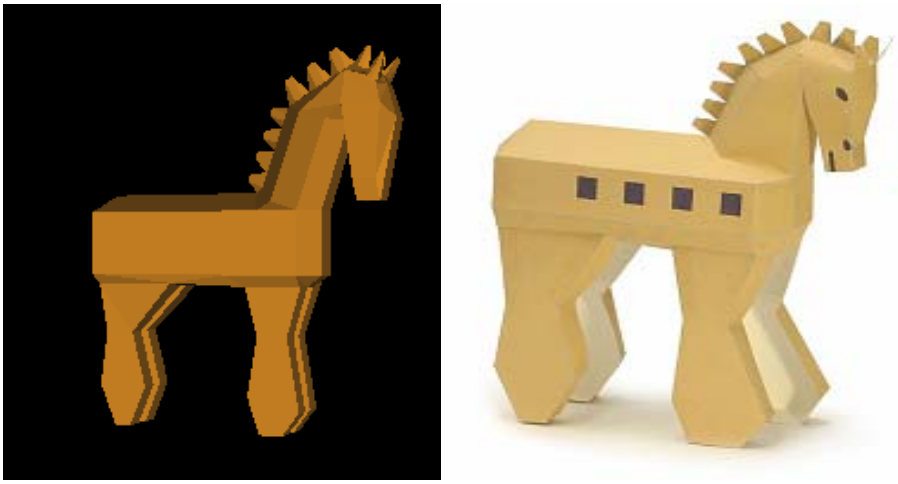


Figure 17. Trojan Horst model (left) and inspirational image of a Peramodel Trojan horse (3D paper model) from http://www.venus.dti.ne.jp/~kpd/jpg/world/trojan.jpg

b. Virus

A computer virus is merely lines of malicious code. Representing a virus in a three-dimensional world with a few lines of text may confuse a user. Instead, the virus modeled after a biological representation of a virus structure. Biologically, viruses take on many different structures. The image that provided inspiration and the resulting model are displayed in Figure 18. The virus model was given a red color to catch the attention of the user.

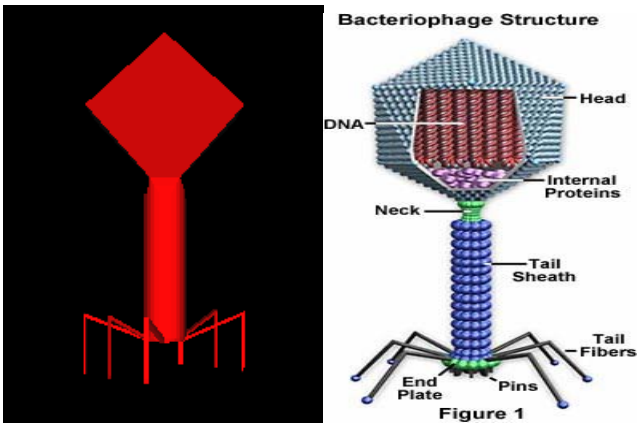


Figure 18. Virus model (left) and the inspirational virus image from the Molecular Expressions™ website at http://www.microscopy.fsu.edu/cells/virus.html March 2007.

c. Trap Door

A trap door is defined as "a mechanism embedded within a system that allows the normal access paths or access checks of a system to be bypassed. This often takes the form of a special password that is hard-coded into the software. It can also take the form of a special diagnostic interface" (Berg, 2005). The model created to represent this attack took the form of an animated wall. An opening appears in the center of the wall and remains open for a short period of time before returning to its original solid state. Figure 19 shows two trap door models, one model with the door closed and the other with the door open. Like the virus model, the trap door model was also given a red color to draw the attention of the user.

Figure 19. Trap door model closed (left) and open (right)

d. Operating System Flaw

An operating system flaw can be thought of as a bug in the operating system code. This bug in the code is born when the operating system is written. Once a flaw is found in the code, it can then be exploited. With this notion of a bug in mind, the model for an operating system flaw was modeled after an insect. The inspirational insect is some sort of beetle. Just as with the trap door model and the virus model, the operating system model was colored red. The model is displayed in Figure 20 below.

Figure 20. Screen shot of an operating system flaw model.

e. Physical Theft

Physical theft is an easy concept for a person to visualize. One can visualize a person stealing a component. Rather than having a person come through the visualization and swipe a component, or make the component randomly disappear, which may be confusing to the user, the decision was made to have a message displayed to the user informing him/her that a particular component was stolen. In case the location of the message does not clearly indicate which component was stolen, an arrow is included with the message and points directly at the stolen component. Figure 21 shows the message displayed to the user and its position in relation to the stolen component. The text remains red for a period of time before briefly fading to white and then back to red. This brief change in color is meant to draw the attention of the user.

Figure 21. Message displayed during a physical theft attack

Each compromise is represented by a different geometric model. These models will appear just to the left of the component that has been compromised. The decision to place the attack model to the left of the component was made because in many cases components might be stacked upon one another and if the model was placed above a component that has components stacked atop it, the model would not be clearly visible. Placing the model just beside the component makes it visible in all cases except when components are placed side by side. It is an uncommon configuration to have components located side by side in CyberCIEGE scenarios.

6. The Attacker

Although an attacker is a physical person, representing the attacker with a model of a real person was not a logical option because not only would the model be complicated, but the model may distract a user from the remainder of the visualization.

After searching the Internet for intimidating figures that could be easily construed as such by a user, it was decided to create a partial person model. The model is basically just a person's head and shoulders and was inspired by a police sketch image from New Zealand. The image served as both an inspiration for the actual geometry of the attacker icon and also serves as the texture used to color the geometry. The resulting textured geometry and the image that provided inspiration are displayed in Figure 22.

Figure 22. Police sketch (right) of a suspect taken from identikit in Fairfax, New Zealand. This image applied to the attacker geometry as a texture (left).

C. INTERACTION

One of the key take-ways mentioned at the end of chapter two was importance of user interaction with a visualization. Users should have the ability to interact with the visualization in a variety of ways. The methods of interaction within NTAV3D include basic navigation capabilities, mouse-over capabilities, and a limited ability to modify the scene.

1. Navigation Capabilities

Navigating through a three-dimensional world can be tricky. Users can become easily disoriented if they are allowed to travel inside solid objects or if the navigation controls are difficult to use. The main navigation controls were modeled after the navigation controls that are currently in place in CyberCIEGE. One reason for doing this is that the navigation controls in place within CyberCIEGE are easy to use and afford the user the freedom to navigate anywhere they desire in the scene. The main reason for

making the navigation controls within the visualization application resemble the controls in CyberCIEGE is so that game players do not have to learn a separate set of controls to navigate within the visualization application. A list of the available "hot keys" and the corresponding actions produced by pressing these keys for both CyberCIEGE and NTAV3D is listed in Table 1. After observing the table, it should be clear that the majority of the main navigational controls utilized in CyberCIEGE are implemented in the visualization in the same or similar manner.

The method for panning the camera side to side and forward and back in is one of the most notable differences. While the panning motion itself is essential, duplicating the CyberCIEGE implementation using off screen mouse movements would have required a large time investment. Investing a large amount of time just to exactly match an implementation style that the user is familiar with was not considered a good investment, when underlying feature could be easily replicated using input from the keyboard. Under this justification, it was decided to replicate the panning motion using selected hot keys.

Because the camera does not automatically change its orientation while zooming or changing elevation, separate controls were added to allow the user to change the camera's orientation. This allows the user to directly control the camera angle while viewing the scene from above or below.

Toggling between sites using only one key was the last useful navigation feature in CyberCIEGE that was not implemented in NTAV3D in the same manner. Instead of using one key, two keys are used to switch between the main site and the remote site. An additional hot key was defined in order to move the view to where the entire scene (both main and remote site) are visible.

NTAV3D		CyberCIEGE	
Key	Action	Key	Action
A	Move camera down	A	Lower the camera
a	Move camera up	a	Raise the camera
Z	Move camera forward (zoom in)	Z	Zoom in
z	Move camera backwards (zoom out)	z	Zoom out
r	Rotate camera clockwise	r	Rotate camera clockwise
t	Rotate camera counterclockwise around	t	Rotate camera counterclockwise
f	Move camera right	u	Iterate through users
g	Move camera left	s	Iterate through support staff
v	Change camera's orientation to look down	m	Iterate through computer components
b	Change camera's orientation to look up	d	Iterate though network devices
H	Move camera to view entire scene and set the pivot point to the center of entire scene	Tab / Shift Tab	Iterate through simple camera positions
h	Move camera in front of main site and set the pivot point to the center of this site	h	Home (in current office)
l	Move camera in front of remote site and set the pivot point to the center of this site	l	Toggle between main office and remote site
Ctrl	Make walls appear	Ctrl	Slows panning
Alt	Make walls disappear	Move cursor off screen	Pans the camera in the direction of the cursor

Table 1. List of "hot keys" for both the visualization and CyberCIEGE

2. Mouse Over Capabilities

All geometry displayed in the scene besides the network links and the walls have mouse over functionality built in. A game player can find out more information (component name, network name, or attack type) about these objects in the scene by placing the mouse over a piece of geometry. When the mouse is over a computer, router, or server, the name of the component will appear directly in front of the component in

large white letters. When the mouse is over a network main line, the name of the network pops up in front of the main line. Placing the mouse on an attack model display the name of the attack above the model. As soon as the mouse is moved off the component, the text will disappear again. The reason for text appearing and disappearing is to keep the scene as clutter free as possible. Having all the component names, network names, and attack types displayed on the screen at once would create a chaotic and confusing scene due to limited screen real estate and numerous elements vying for space. Messages would overlap forcing the user to decipher the alphabet soup displayed on the screen. This type of burden on the user would make it hard for the user to extract the information being presented. With the current mouse-over implementation, only one message will be displayed at any given moment because the mouse can only be over a single component. This eliminates the possibility of overlapping text and makes the message easy for the user to read.

Attempting to place the mouse over a network link that has a width of one pixel would be extremely difficult and all necessary information (which component and which network the link is connected to is apparent from the scene). For these reasons, mouse-over functionality was not build into network links.

3. Scene Modification Capabilities

The only modification of the scene that can be performed by the user is toggling the walls from visible to invisible and vice versa. This capability is necessary because the walls interfere with the mouse-over capability of components obscured by them. In the absence of walls, the user is able to navigate more freely and access the mouse-over features of components that were previously obscured by the walls. Turning the walls back on restores visual familiarity and the spatial reference frame provided by the walls' presence. It should be noted that in order to turn the walls back on the user must press the Alt key twice in succession. The precise reason for this is not known, but this issue is explored in more depth during the section on evaluation of Xj3D in chapter six.

D. ATTACK ANIMATIONS

Animation is used to both grab the user's attention and to show data flow through the network. Animation is only used during an attack. The animation used for component attacks is different from the animation used during an asset attack.

1. Component Compromises

As mentioned earlier in this chapter, component compromise models are placed just to the left of the component that has been compromised. Each of these models is animated so as to attract the attention of the user. The Trojan horse, virus, and operating system flaw models all have animated scales. This means that each model periodically gets larger then smaller. The trap door model looks like a solid wall for a few moments, and then a hole begins to open in the center of the wall. This hole remains open for a few second before closing, returning the wall back to its original solid form. The message in the physical attack changes color. It cycles between red and white, spending the majority of the time as red text.

Although availability attacks are not explicitly represented in NTAV3D, users are able to determine when these attacks have occurred by observing the presence of component compromise models and the absence of an asset attack animation.

2. Asset Attacks: Integrity vs. Secrecy

In order to clearly represent the flow of traffic through the network during an asset attack, information in the form of a cone was animated to travel through the network. The animations for integrity attacks and secrecy attacks have a couple key differences. The main, and most noticeable, difference is in the direction of traffic flow. In an integrity attack information flows from the attacker to the asset. In order to convey the direction of flow a cone travels through the network to present the path taken by the information. Throughout the cone's journey it say oriented such that the tip of the cone points in the direction of travel. In this way, the direction of travel can be ascertained from a static screen shot. During a secrecy attack, the flow is from the asset to the attacker.

Besides the direction of flow, the name of the asset attacked is displayed above the cone during a secrecy attack, while nothing is displayed during an integrity attack. The reason nothing is displayed during an integrity attack is because nothing is known about the information being sent by the attacker. By not directly informing the player which asset is being attacked during an integrity attack, the player must observe which component is being attack and reference the assets contained within the component to determine which asset is being compromised. When a component contains multiple assets, the component reference method fails. Determining how and where to present information about which asset is being attacked during an integrity attack is a topic for future work.

One final way in which the user can distinguish between integrity and secrecy attacks is by the color of the cone. During a secrecy attack, the cone will be green, while the cone will appear red during an integrity attack. The choice in coloring was made solely to reinforce the fact that these two attacks are different.

E. ROUTING NETWORK LINKS

Because the positions of components like computers, routers, and servers is important and these positions are directly specified by CyberCIEGE, a unique routing scheme needed to be devised in order to route links between components and network main lines.

As shown in Figure 15, all network main lines are located at the front edge (largest z value) of the main site, but at different depths. The depth of a network main line is dependent on the order in which it is specified in the input file. The first one specified is at the shallowest depth below the floor, with the following network main lines at incrementally deeper depths. Placing all the network main lines at one edge of a site eliminated the possibility of a network link routing through a network main line as a link routes down to the corresponding network depth because none of the main lines will be directly under a component.

In order to minimize ambiguity with connections between network links, a clear presentation scheme was developed in order to eliminate different colored lines crossing

(lines from different networks). To accomplish this task, a basic routing scheme was developed where links route out from a component in the x-direction, then down through the floor and over to the network main lines at the front of the site. The routing distance in the x-direction eliminates the crossing of network links as they route from components to network main lines by testing the "drop point" against already established network links. The drop point is the point at which the network link is dropped through the floor. If the drop point matches one from an existing link from a different network, then the drop point is incremented and tested again. This process is repeated until drop point is found that satisfies the test criterion. A clear presentation of routing can be found in Figure 23.

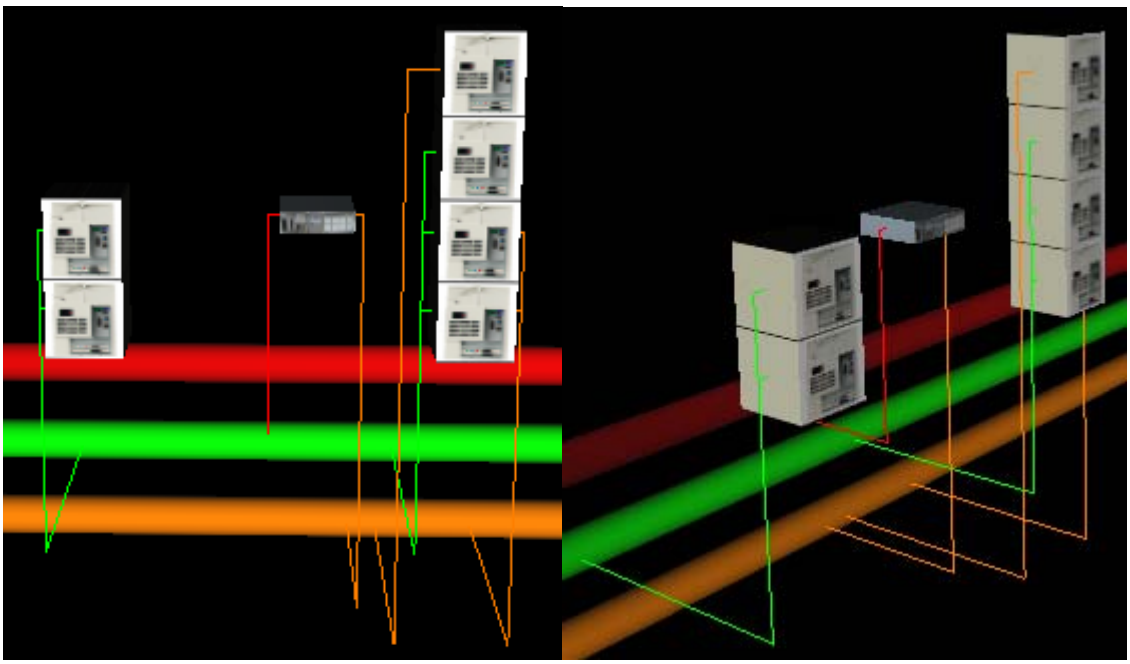


Figure 23. Presentation of routing network links from an example scene viewed from two different angles.

Figure 23 displays two different views of the same scene from different angles to showcase how network links on different networks (differentiated by colors) do not cross. This example scene has many components packed in a small area and without proper routing would be confusing to a game player due to crossing lines. Only network links of the same color, like the green lines routing out from the stacked servers, route down to their networks with the *same* drop point. This greatly reduces the amount of clutter in the

scene and emphasizes the interconnections between links on the same network. This method of presentation was not implemented to make *all* network connections from a particular network use the same drop point when routing from stacked components. If this were the case, the orange line coming out of the top server would meet up with the orange lines on the other side of the stacked servers. This inconsistent presentation is due to a coding implementation, but with some additional work, the routing scheme within the NTAV3D application can be refined to include a single drop point for each network connected to components in a stacked configuration.

V. APPLICATION IMPLEMENTATION

A. JAVA APPLICATION ARCHITECTURE

1. Xj3D (SAI) Scene Access Interface Implementation

Three dimensional scenes are not something new, but creating them by utilizing only the Java-based Scene Access Interface (SAI) of the Xj3D tool kit is. This is important because NTAV3D is one of the first applications solely developed utilizing the Xj3D SAI. A limited number of examples describing how to use the SAI can be found at the Xj3D website, and all of the examples rely on outside files for the actual creation of geometry. The sections to follow outline both the basics for how geometry, user interaction, and animations were actually created within NTAV3D and also explain some of the programming syntax involved.

The basic scene graph structure that is utilized within the Xj3D's SAI is the same as within X3D. A simplified scene for the creation of the transparent walls within NTAV3D can be found in Figure 24. All the nodes used within the scene are children of the Scene node. In addition to the basic parent-child relationships displayed in the scene graph, Figure 24 also notes which fields are important within each node, and displays a routing diagram that is denoted by the hashed lines. The various components of the scene graph presented in Figure 24 will be explained over the next three subsections.

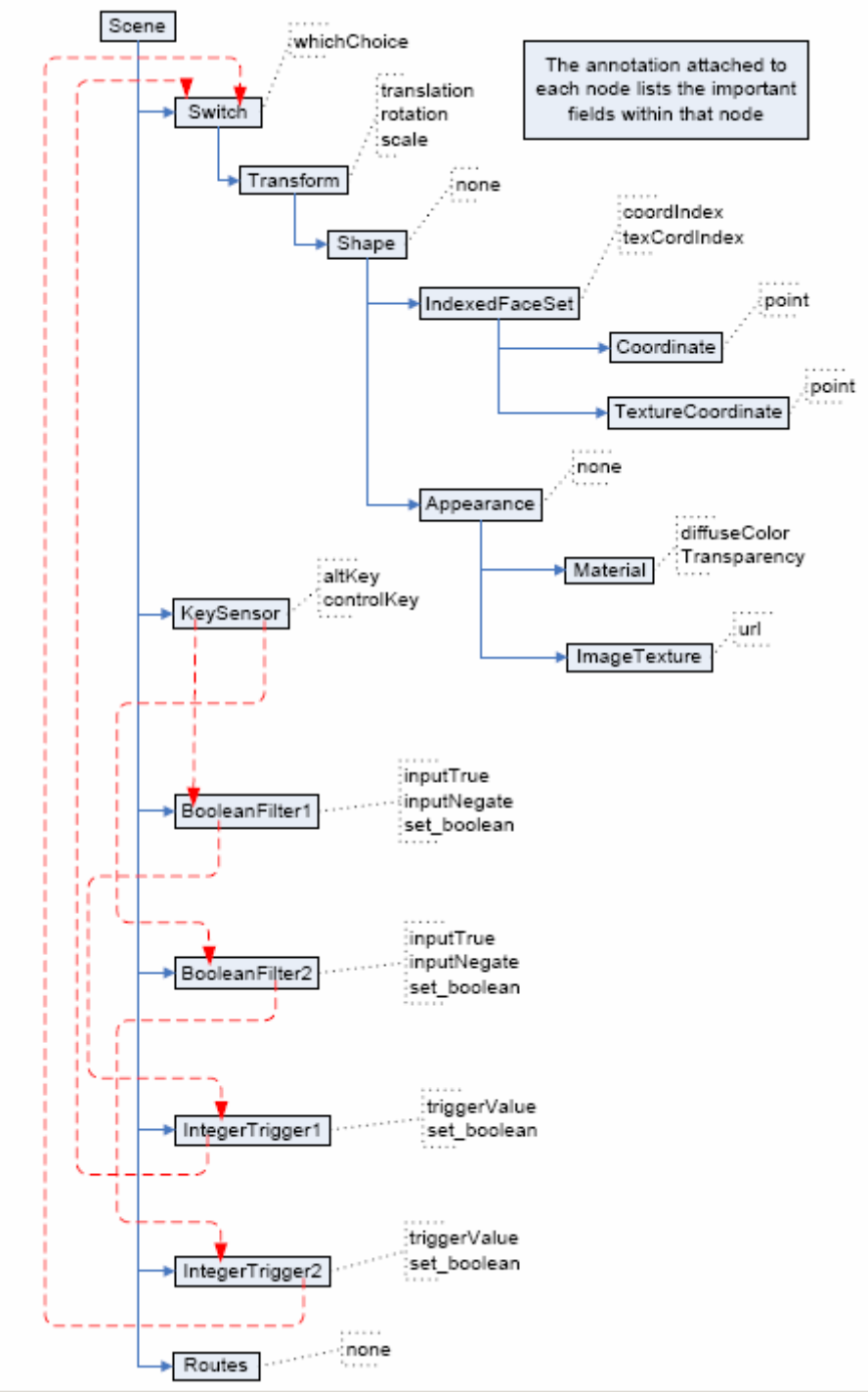


Figure 24. A simplified scene graph for creating walls within NTAV3D that notes important fields for nodes and includes a routing diagram

a. Geometries Used in NTAV3D

As mentioned in Chapter IV, physical objects within NTAV3D are represented by models resembling these objects. The majority of the geometry visible in

NTAV3D was created using indexed face sets and indexed line sets. In addition to these two types of geometry, cones and extrusions were used.

(1) Xj3D Geometries Explained

Indexed face sets are one method for creating geometry. They are comprised of a set of polygonal faces. The first step in creating an indexed face set is to define the list of vertices that describe the structure of the object in the "points" field of the *Coordinate* node attached as a child. The next step is to delineate which vertices make up which face in the "coordIndex" field of the *IndexedFaceSet* node. This is accomplished by referencing the indices of vertices in the list. A single vertex can be a part of several polygonal faces. This framework for creating objects is efficient by way of reusing vertices, and versatile in that any solid object can be modeled. Indexed face sets were used to model the walls, computers, routers, servers, the Internet cloud, all the component compromise models (except physical theft), and the attacker icon in NTAV3D. Figure 24 illustrates the scene graph structure involved in creating an indexed face set.

Indexed line sets are very similar to indexed face sets. They too are comprised of a list of vertices, but instead of a list that creates polygonal faces, an indexed line set has a list that specifies the order in which vertices are applied to create a segmented line. Indexed lines sets were used to create the various network links seen throughout scenes in NTAV3D. These lines have a unique property of always having a thickness of one pixel regardless of how close or distant the line is within the scene. Replacing the *IndexedFaceSet* node with an *IndexedLineSet* node is the only change to the scene graph in Figure 24 needed to create an indexed line set.

Cones are one of the predefined simple geometries available in Xj3D. The only information necessary to create a cone is a height and a radius. Within NTAV3D, cones were used to represent data. One only need substitute the *IndexedFaceSet* node and its children in Figure 24 with a *Cone* node in order to create a cone.

Extrusions are the last method used to create geometry within NTAV3D. An extrusion is defined by a two-dimensional cross section and a spine. The cross section defines the prominent structure of the object. An extrusion is created by repeating the cross-section along the vertices defined in the spine. The spine can also be thought of as a set of controls points for controlling the smoothness between cross-sectional slices. Extrusions were used in NTAV3D to model the network main lines. The cross-section was a circle, and the spine consisted of a beginning point and an end point. This created a three-dimensional pipe. An extra point in the spine was necessary in the case when a network main line connected to the Internet cloud. This created a smooth pipe with a single bend in it as illustrated by the red pipe in Figure 29 (page 79). Substituting the *IndexedFaceSet* node Figure 24 with an *Extrusion* node is all that is required to create an extrusion.

(2) Examining the Java Syntax

Understanding the relationships within the scene graph in Figure 24 and knowing which node fields are important make coding easier. This section outlines how the walls were created within NTAV3D by examining code snippets from the **WallCreator** class, which can be found in its entirety in Appendix B.

The pattern for creating any node within the scene is shown in the first line of the code snippet below.

```
X3DNode tr = scene.createNode("Transform");
MFNode tran_children = (MFNode) tr.getField("children");
tran_children.clear();
```

In the first line, a *Transform* node is created and added to the scene by providing the **createNode()** method with the string "Transform." A list of other nodes used within NTAV3D can be found in Appendix A. Because a node is created by specifying the name of the node via a string, the spelling must be exact or a run-time error will occur. A *Transform* node is a type of grouping node, so it was necessary to be able to add children to this node. The second and third lines of code create access to the "children" field and clear this field in preparation for adding children. The next step to create the actual geometry.

All geometry nodes are created as a child of separate *Shape* nodes. Creating the *Shape*, *IndexedFaceSet*, and *Coordinate* nodes is accomplished in the same manner that the *Transform* node was created.

```
X3DNode shape = scene.createNode("Shape");
X3DNode box = scene.createNode("IndexedFaceSet");
X3DNode coordinate = scene.createNode("Coordinate");
```

The method for defining the vertices that comprise the indexed face set is not as straightforward as adding the node. A special relationship is formed between the parent node (*IndexedFaceSet*) and the child node (*Coordinate*) using a SFNode. Through this relationship, the values set in the "points" field of the *Coordinate* node are assigned to the "coord" field of the *IndexedFaceSet* node.

```
SFNode line_coord = (SFNode) (box.getField("coord"));
MFVec3f point_value = (MFVec3f) (coordinate.getField("point"));
point_value.setValue(8, new float[] { this.corner1[0], -1,
this.corner1[1],. . . this.corner1[0], 3, this.corner2[1], });
line_coord.setValue(coordinate);
```

Setting the value of the "coordIndex" field of the *IndexedFaceSet* node is much more straightforward since this field exists within the *IndexedFaceSet* node. This field references the vertices defined in the "points" field of the *Coordinate* node by using indices. Individual polygon faces are separated by values of negative one. Because some browsers implement indexed face sets in different manners it is a good practice to completely close faces by ending with the same point that began the face (first specified face is ***8***, 9, 5, 4, ***8***, -1).

```
MFInt32 coord_index = (MFInt32) (box.getField("coordIndex"));
coord_index.setValue(30, new int[] { 8, 9, 5, 4, 8, -1, 9, 10, 6,
5, 9, -1, 10, 11, 7, 6, 10, -1, 11, 8, 4, 7, 11, -1, 4, 7, 6, 5,
4, -1, });
```

Once the geometry is defined, it must be added to the scene. This is accomplished using another special relationship node to assign the *IndexedFaceSet* node as the "geometry" field value for the *Shape* node. The *Shape* node is then appended as a child of the *Transform* node. The *Transform* node becomes a child of the *Switch* node, which is then added to the scene.

```
SFNode shape_geometry = (SFNode) (shape.getField("geometry"));
shape_geometry.setValue(box);
tran_children.append(shape);
switch_children.append(tr);
scene.addRootNode(switchNode);
```

Both the direct and indirect methods for adding nodes to the scene are illustrated in the previous code snippet. The direct method explicitly adds the node to the scene. The indirect method involves appending nodes as children of nodes that are directly added to the scene.

b. Texturing

(1) What is a Texture and What is it Used for?

A texture is an image that is put on an object to color it. Texturing is done on a per fragment basis, whereas assigning color is done on a per vertex basis. Using a texture allows one to explicitly assign colors per fragment using texture coordinates (texels) as opposed to interpolating color values between vertices. Similar results could be achieve by manually assigning colors to vertices, but a much larger number of vertices for the same geometry is required. Texturing is an extremely common and widely used method in the gaming industry (both console and computer games).

(2) How are Textures Placed on Objects?

The example code that follows is taken from the AttackerIcon class. The example code in this subsection shows how the rectangular image of the attacker in Figure 22 (page 41) was mapped to the oddly shaped geometry in the figure. An *Appearance* node and an *ImageTexture* node do the bulk of the work in assigning a texture to an object.

```
X3DNode appearance = scene.createNode("Appearance");
X3DNode image_texture = scene.createNode("ImageTexture");
```

As with the creation of an indexed face set, special relationships are utilized while assigning values from a child node to a parent node. The actual name and location of the image file used as the texture is specified in the "url" field of the *ImageTexture* node.

```
SFNode shape_appearance = (SFNode) (shape.getField("appearance"));
SFNode appear_texture = (SFNode) (appearance.getField("texture"));
MFString textureURL = (MFString) (image_texture.getField("url"));
textureURL.setValue(1, new String[] { "images.jpg" });
appear_texture.setValue(image_texture);
```

In the forth line the file name is specified in the **setValue**() method by using a string array and its length. A string array is used so that multiple paths to the same image file can be specified, and if the first path fails, the next path in the array list will be tried. This provides alternative URLs for redundancy when referencing images on the web.

This next snippet of code shows the steps necessary to map a rectangular image to an irregular polygonal face. The "texCoordIndex" field of the *IndexedFaceSet* node is set in the same manner that the "coordIndex" field is set.

```
MFInt32 texCoord_index = (MFInt32)
        (attacker.getField("texCoordIndex"));
texCoord_index.setValue(textCoordIndex.length, textCoordIndex);
```

As seen in Figure 24, a *TextureCoordinate* node is created as a child of the *IndexedFaceSet* node. Texels are defined in the "points" field of the *TextureCoordinate* node in much the same way that vertices are defined in a *Coordinate* node. Using the special parent-child relationship node, these texels are assigned back to the "texCoord" field of the *IndexedFaceSet* node. Once these values are assigned back to this field, they can be mapped to vertices on the indexed face set.

```
X3DNode textureCoord = scene.createNode("TextureCoordinate");
SFNode text_coord = (SFNode) (attacker.getField("texCoord"));
MFVec2f textCoord_value = (MFVec2f)
        (textureCoord.getField("point"));
textCoord_value.setValue(textureCoordinates.length / 2,
        textureCoordinates);
text_coord.setValue(textureCoord);
```

c. User Interaction

(1) Types of User Interaction

The two avenues by which the user can interact with scenes created by NTAV3D are the keyboard and the mouse. As described in Chapter IV, the keyboard is used to move around within the scene and to turn walls on and off. The mouse is used to uncover additional information about objects (i.e. the components name, the name of the zone, or the name of a network) by placing the mouse over a given component.

(2) Creating Interaction with the Xj3D SAI

A *Switch* node is a type of grouping node that renders one or none of its renderable children. Which child is rendered is controlled by the field "whichChoice," and by default, it is set to the first child. In the following code snippet, a *Switch* node is created and set to render its first child. No children are rendered by setting the value of the field "whichChoice" to minus one.

```
X3DNode switchNode = scene.createNode("Switch");
MFNode switch_children = (MFNode) switchNode.getField("children");
switch_children.clear();
SFInt32 whichChoice = (SFInt32)
switchNode.getField("whichChoice");
whichChoice.setValue(0);
```

The first step in implementing the mouse-over feature to geometry in the scene was to add a *TouchSensor* node. Touch sensors receive input from the mouse. They know where the mouse pointer is located and whether a click is registered. Adding a *TouchSensor* node is the same as adding any other node, but it needs to be added to the scene graph such that it is at or above the level of the desired geometry. In the code snippet below, the *TouchSensor* is added to the *Transform* node as a child, which means that it will be associated with any geometry nodes that are children of that *Transform* node.

```
X3DNode touchSensor = (X3DNode) scene.createNode("TouchSensor");
scene.addRootNode(touchSensor);
tran_children.append(touchSensor);
```

Once a touch sensor is linked to geometry, it is able to determine when the cursor is placed over or clicked upon a geometry linked to it. The next step is to introduce the necessary logic components and route information to the necessary nodes. Only one *BooleanFilter* node is necessary to create the mouse-over feature, while two are needed to toggle the walls on and off. A *BooleanFilter* node takes in a Boolean value and outputs a Boolean value. The filtering aspect of this node comes from sending out a value only when specific Boolean input is received, or the Boolean input can be negated and sent out. The following code snippet shows the creation and addition of a *BooleanFilter* node to the scene.

```
X3DNode bf3 = scene.createNode("BooleanFilter");
scene.addRootNode(bf3);
```

Once the *BooleanFilter* is in place, a series of *IntegerTrigger* nodes is necessary to convert Boolean inputs into integer outputs. The "integerKey" field is where the desired output integer value is specified. Only one integer value can be specified for each *IntegerTrigger* node. An *IntegerTrigger* node only outputs the integer value when a Boolean true event is received. Two *IntegerTrigger* nodes are created and assigned different output values before being added to the scene in the following code snippet.

```
X3DNode triggerTextOn = scene.createNode("IntegerTrigger");
SFInt32 integerKeyOnText = (SFInt32) triggerTextOn
         .getField("integerKey");
integerKeyOnText.setValue(0);
scene.addRootNode(triggerTextOn);

X3DNode triggerTextOff = scene.createNode("IntegerTrigger");
SFInt32 integerKeyOffText = (SFInt32) triggerTextOff
         .getField("integerKey");
integerKeyOffText.setValue(-1);
scene.addRootNode(triggerTextOff);
```

The last step is to route information from one node to another to create the desired interaction. *Route* nodes are used to send information from one node to another. A route sends values from an output field in the source node to an input field in the destination node. *Routes* are depicted in Figure 24 by hashed red lines with a single arrow head conveying which node is the source and which is the destination. *Route* nodes are a vital component of any interaction or animation that occurs using X3D, Xj3D, or VRML97.

```
scene.addRoute(touchSensor, "isOver", bf3, "set_boolean");
scene.addRoute(bf3, "inputTrue", triggerTextOn, "set_boolean");
scene.addRoute(bf3, "inputFalse", triggerTextOff, "set_boolean");
scene.addRoute(triggerTextOn, "triggerValue", switchNodeText,
        "set_whichChoice");
scene.addRoute(triggerTextOff, "triggerValue", switchNodeText,
        "set_whichChoice");
```

Matching field types is vital during routing process in order for things to work. Boolean values must be routed to Boolean fields and integer values must be routed to integer fields.

The *BooleanFilter* is the intermediary for Boolean values between the *TouchSensor* and the *IntegerTtriggers*. Without the *BooleanFilter*, Boolean values would be routed from the *TouchSensor* to both *IntegerTriggers* at the same time resulting

in conflicting commands. Both *IntegerTriggers* would attempt to send integer values at the same time, and the last integer received would be the one to influence which node would be rendered by the *Switch* node. By having the *IntegerTriggers* receive values from the Boolean filter, only one *IntegerTrigger* receives a value during an event and conflicts are avoided while setting the rendering choice of switch node.

d. Animation

(1) Nodes Vital to Animation

The three nodes that make animation possible are the *TimeSensor*, an *Interpolator*, and *Route* nodes. The *TimeSensor* node acts as a clock. Fields within the *TimeSensor* node allow one to specify whether the animation will be looped, and the duration of the animation. A *TimeSensor* also keeps track of the fraction of time that has elapsed during the animation. This faction is useful for the *Interpolator* node. The fraction of time elapsed tells the *Interpolator* node how much to interpolate the values within the *Interpolator* node. The *Route* nodes are used to connect the *TimeSensor* to the Interpolator, and the *Interpolator* to the component being animated. The lines of code to follow in the next subsection were taken from the PhysicalAttack class. The entire class can be found in Appendix B.

(2) How to Create Animation using Xj3D SAI

The first step to creating an animation is to create the *TimeSensor*. Once the *TimeSensor* is created its fields can be modified to enable a looped animation and the cycle interval can be set.

```
X3DNode timeSensor = scene.createNode("TimeSensor");
SFBool loop = (SFBool) timeSensor.getField("loop");
loop.setValue(true);
SFTime cycleInterval = (SFTime)
timeSensor.getField("cycleInterval");
cycleInterval.setValue(2.0);
scene.addRootNode(timeSensor);
```

The next element necessary to create an animation is to create the *Interpolator* node or nodes. The two fields of importance within an *Interpolator* node are "keys" and "keyValues." The first is a list of instants in time specified by floating point

values between 1.0 and 0.0. The second field lists the state for each instant in time listed in the "keys" field. The code snippet below shows a *PositionInterpolator* node being created and the "keys" and "keyValues" fields being set.

```
float[] keys = { 0, .25f, .75f, 1 };
float[] keyValues = { 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1 };
X3DNode pi = scene.createNode("PositionInterpolator");
MFFloat key = (MFFloat) pi.getField("key");
key.setValue(keys.length, keys);
MFVec3f keyValue = (MFVec3f) pi.getField("keyValue");
keyValue.setValue((keyValues.length) / 3, keyValues);
scene.addRootNode(pi);
```

As with user interactions, the last step is to connect *Route* nodes between source node fields and destination node fields. The routing pattern shown in the code snippet below is the pattern used when employing an *Interpolator* node.

```
scene.addRoute(timeSensor, "fraction_changed", pi,
        "set_fraction");

scene.addRoute(pi, "value_changed", material2, "diffuseColor");
```

2. Java Class Structure

The basic programmatic flow of the application is depicted in Figure 25 on the following page. This diagram depicts the steps of execution each time NTAV3D is launched, and allows one to gain a better understanding of how the different parts of NTAV3D's code interact with each other. For a full UML (Unified Modeling Language) class diagram, which breaks down all of the Java classes and their elements, see Appendix C.

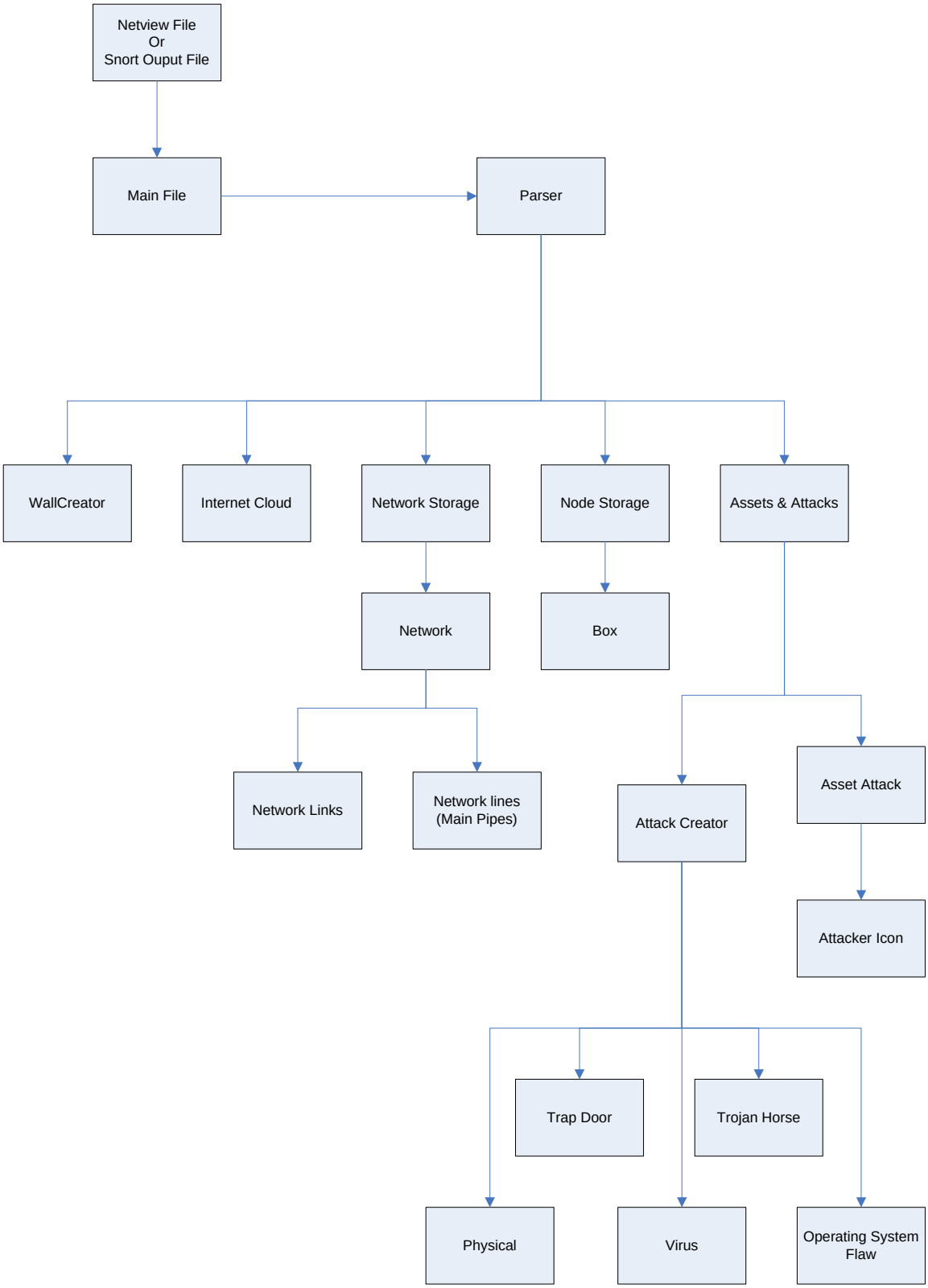


Figure 25. Application Flow Chart

B. XML PROCESSING

1. XML in NTAV3D

As mentioned in Chapter III, XML was used to exchange data between outside applications and NTAV3D. In this way, XML functions as a kind of "middleware" to the application. The following sections will serve to describe how XML is utilized in NTAV3D. The structure of the XML file will be discussed, including what data was included and why. Then, the way NTAV3D actually processes the XML file will be described. Knowledge of how NTAV3D handles XML is helpful to understand how the program processes data, and will also facilitate a better understanding of how it can use data from other programs (which will be addressed in the next section).

2. File Structure

To understand the structure of the XML file NTAV3D receives from CyberCIEGE, Appendix D contains the CyberCIEGE document type definition, which specifies the format of the XML file output from CyberCIEGE. Appendix D also contains an example XML file that was actually output from CyberCIEGE. This is the same example scene depicted in Figure 29. The CyberCIEGE XML file's structure is one that starts at a higher level of detail about the networks, and then continually drills down to include more details about the networks and their components. The file starts with high level data, such as which networks are currently involved in the game, and details about them such as their name, and the color that they are mapped to in the game. NTAV3D maintains this color mapping when displaying networks in an effort to help player orientation. The next part of the XML file includes definitions of each network zone (which corresponds to the previously mentioned transparent walls displayed), and the components within each zone. Each component has such attributes as the type of component, the name given in the game, the networks the component is connected to, and any data assets that may exist on the device. An example of this data in XML is shown below.

```
<network>
        <networkName>Offsite LAN</networkName>
        <color>0xFFFFFF00</color>
</network>
<mainSite>
```

```
        <zoneName>Entire Office</zoneName>
        <upperLeft><x>33</x><z>49</z></upperLeft>
        <lowerRight><x>56</x><z>31</z></lowerRight>
        <component>
                <componentName>Joe_ws</componentName>
                <componentBase>Blato Desktop Select</componentBase>
                <location><x>35</x><y>1</y><z>36</z></location>
                <networkName>leased</networkName>
                <networkName>Lan1</networkName>
                <assetName>Plans</assetName>
        </component>. . .
</mainSite>
```

Finally, the XML file can contain information on the kind of network attack occurring on or over the network. This provides NTAV3D with a way of visualizing the attacks that occur in CyberCIEGE. This information will only be included in the file at the time of an attack. See the next section for a discussion of this attack data.

3. CyberCIEGE XML Information

a. Topology Data

During the course of NTAV3D development, changes were made continually to the structure of the CyberCIEGE Document Type Definition (DTD) so that information provided to NTAV3D is more useful. For example, to show the network topology and general office structure, the physical location of components, and office "walls," or zones, was needed. For easier topology and data display, the names of each component and the names of the networks it is connected to was needed. Therefore, for each zone and component, location data was added to the XML file specification. As will be discussed later, providing a physical location for components in a real network is quite difficult, so including location data in the CyberCIEGE file helped to eliminate a layer of complexity. CyberCIEGE already was storing location data from the "office view", it only made sense to provide that information to NTAV3D. Another benefit of this is that it allowed the mappings of where objects were located in the "office view" to be similar in NTAV3D. Again, this helps aid gameplayer orientation.

b. Attack Data

Another crucial area of data that was needed from CyberCIEGE was that relating to network attacks. This is vital for NTAV3D, as it allows it to visualize the attacks that occur in the game, as opposed to the game's current generic movie clips. After conducting the research discussed previously, the details of a network attack to include in the CyberCIEGE XML file was decided. First, NTAV3D has to know which network component or components were compromised. Next, the specific asset that was on the components must be known. Once this data is outlined, "segment" data is needed in order to show the information flow resulting from the attack. This information will determine the route the attack followed by providing the networks and networking components that the attack passed through. Once these segments are known, NTAV3D then uses an algorithm to determine the specific path of the attack, and thereby show an animated cone to represent data flow. Finally, CyberCIEGE outputs the type of attack occurring (secrecy, integrity, Trojan horse etc.), and also the type of attacker. This information allows NTAV3D to display its icons specific to each attack, so that the gameplayer is given context of which kind of attack is occurring. An example of this data in XML format is shown below.

```
<attack>
        <assetName>Basic Research</assetName>
        <policy>Integrity</policy>
        <segment><Network>Internal LAN1</Network><componentName>Bit
        Flipper_3</componentName></segment>
        <segment><Network>Internal LAN2</Network><componentName>Five Inches of
        Asbestos_4</componentName></segment>
        <segment><Network>Internet</Network></segment>
        <attacker>Attacker</attacker>
</attack>
```

By parsing this data, NTAV3D has the information needed to depict a network attack. Figure 26 on the following page shows a screenshot of a running example of a network attack in NTAV3D.

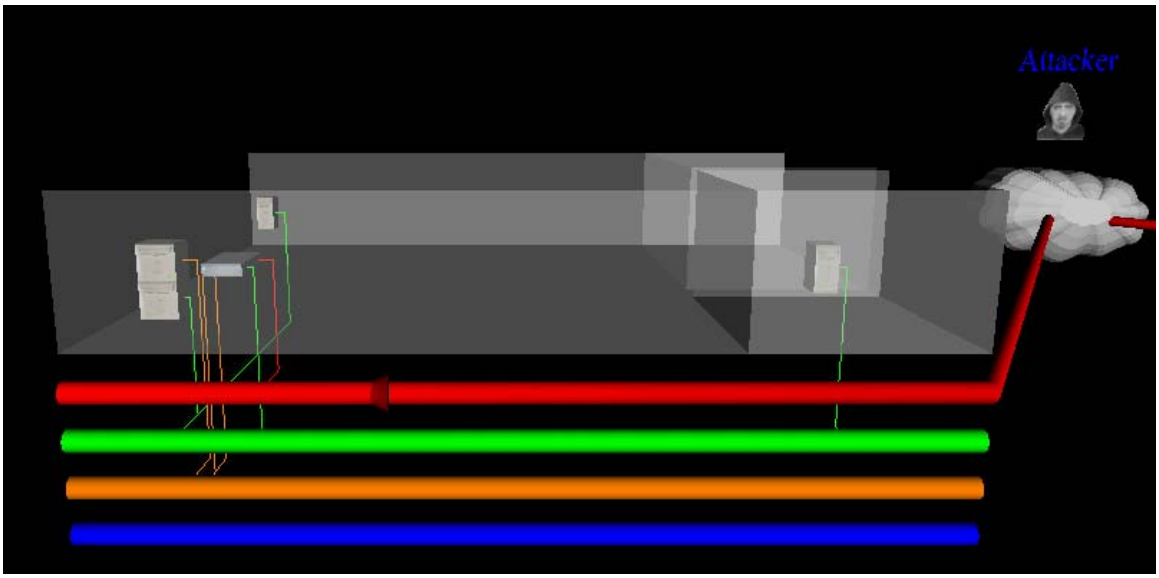


Figure 26. Screenshot of a running example of a network attack in NTAV3D

Note the attacker icon in the figure, as well as the cone to depict data flow seen traveling along the red network (in this image it is directly below the foremost transparent zone wall and moving left). Also, while CyberCIEGE does not currently export data about component compromises or asset attacks, the structure and data content will be similar to that above, and it is expected NTAV3D can display them.

4. XML System Implementation

At the time NTAV3D is launched, it looks for an XML file, whose file system path is specified at run-time as a startup parameter. Once the file is located, the Apache Xerces XML parser is utilized to parse the file. Some of the code behind this is available as an excerpt (for brevity) in Appendix B. The parser then searches for specific XML nodes specified in the Java code, and creates an internal list of the node, and the sub-elements contained in it. For example, it will look for all "component" nodes, and then store a list of the "componentName," componentBase," etc. for each node. Then, the NTAV3D code loops through each list, extracting information as it goes, or taking actions in the Xj3D scene, such as creating a new server object, or constructing walls. An example of this kind of action is in the code below

```
Element netNameElmnt = (Element) netFstNode;
NodeList netNmElmntLst = netNameElmnt
.getElementsByTagName("networkName");
```

```
Element netNmElmnt = (Element) netNmElmntLst.item(0);
NodeList netName = netNmElmnt.getChildNodes();
networkStorage.getNetworkArrEl(holdnet).setNetworkName(
(netName.item(0)).getNodeValue());
```

This code is going through the already defined "Network" list and looking for the element known as "networkName." When it finds this, it gets the value and stores it within the NTAV3D array (a Java programming structure for holding data) known as "networkStorage." This process is then continued for all of the nodes in the XML file.

C. NETWORK TOOL INTERACTION

1. Why Work With Network Tools?

Some intrusion detection systems, such as Snort, can detect attacks automatically and in real-time. Therefore, a valid question is why bother displaying a network attack to a human operator at all? Why not just automatically remove the offending packets from the network and be done? By providing a visual depiction of the attack, and especially one that is interactive and in three dimensions, the objective is to provide the network security professional with increased situational awareness (the need for studies to prove this situational awareness enhancement is addressed in Chapter VI). By being able to "see" the attack happening, a somewhat abstract idea of unseen electronic attack can be put into a more usable context, hopefully one that the security operator can use to his or her advantage to repel future attacks.

2. Tool Output

However, allowing NTAV3D to work with other network analyzers and intrusion detection systems, such as the aforementioned Wireshark or Snort, is a much more complicated problem than that posed by CyberCIEGE. This is mainly due the fact that these network tools are showing the information from a "real" network, instead of the "scripted" one shown in CyberCIEGE. Whereas CyberCIEGE can tell NTAV3D where certain components etc. are located, to discover the physical location of a network node using a program like Wireshark or Snort can be extremely difficult.

Indeed, network topology generation could be a completely separate and independent thesis topic. Along these lines, there are additional differences between the

kinds of data CyberCIEGE can provide, and that which a network tool can. Table 2 below shows a comparison of the kinds of data NTAV3D displays of a CyberCIEGE "office", and the data network tools can provide. The table also shows the difficulty (as rated by the authors) that it would take to obtain the data from a network tool. In some cases, such as office zone names, a network tool would not normally provide that kind of information, but it is included in the table to show the details included by CyberCIEGE that would probably not be able to be included via a network tool.

Data Type	CyberCIEGE	Network Tool (Wireshark Snort, etc.)	Difficulty of Obtaining Data From Network Tool
Physical Office Zone Names & Locations	X	--	6
Network topology	X	X (with topology generator)	5
Network attack origin	X	--	5
Specific asset compromised	X	X	5
Path of network attack	X	X	5
Network Attack Type (Trojan horse, etc.)	X	X	4
Assets stored on component	X	X	4
Network Names	X	X	4
Detect live network attacks	--	X	3
Networks component belongs to	X	X	3
Component types (server, computer, etc.)	X	X	3
Component names	X	X	2
Log real network data packets in real-time	--	X	1

Table 2. Table comparing CyberCIEGE data to network tool data, and the ease of obtaining said data from a network tool. Difficulty is from 1-6, with 1 being easy, 5 being extremely difficult, 3 being moderately difficult, and 6 being basically impossible

As can be seen, some of the most difficult aspects of using a network tool for network visualization involve extracting network topology, and determining data flow. NTAV3D cannot solve these problems, nor is it meant to. NTAV3D has the capability to visualize this information, but actually deriving this data is outside the scope of this thesis.

Representing network attacks in almost "real-time" is quite a challenging aspect of network security. Snort, as an example, can log packets of network data, and attempt to alert network security professionals of possible intrusion attempts. Using the methodology detailed in the next section, it can, with future work, be possible to represent this data in NTAV3D, thus providing a visual context for how data is flowing, and nodes that are compromised, which would be quite useful. For example, if Snort were to detect a Trojan horse program trying to send unauthorized data, NTAV3D could display exactly where that node is, and what internal networks the data is traveling through. Again, however, NTAV3D is only meant to visualize such data, not discover and derive it itself. The next section explains how NTAV3D can be utilized in such a role to provide this type of visualization.

3. Proposed NTAV3D Methodology

a. Data Manipulation

In order to provide visualization capabilities to network protocol analyzers or intrusion detection software, the thesis application could employ its already existing XML parser. This will require the help of outside tools however; as again, NTAV3D is not a stand-alone analysis program. The raw output of such network analysis or intrusion detection programs as Wireshark or Snort could be fed to outside topology generators (such as those discussed in Chapter II). These generators could then be utilized to create topology, while at the same time, the results of the network tools' analysis could be used to determine segments of attack, which could then be provided to NTAV3D.

However, this is still not an easy task. For example, as mentioned above, it is extremely difficult to identify distinct networks from the raw data that a tool such as Snort provides. Thus, a separate network analysis would be needed for each physical

network. This data would then have to be collated together, allowing for the discovery of distinct networks within the data. Furthermore, attempting to collate interconnecting networks with different security statuses could prove troublesome. An example could be trying to collate the Department of Defense's secure SIPRNET with the normal NIPRNET. While NTAV3D itself would not be a potential vulnerability, since it is only providing the visualization, whatever tools are used to try to connect packet flow between the networks (e.g., to transfer NTAV3D data from the NIPRNET to the SIPRNET for collation) could serve as a gateway for attacks.

As mentioned above, obtaining this data is quite difficult. Thus, a possible solution as relates to NTAV3D could be to allow users of the tool to manually associate and provide data about their networks in a similar way that CyberCIEGE does. In this case, a user of NTAV3D may already have a good idea of his or her network's topology, and just wants to use NTAV3D's ability to show three dimensional topology or depict attacks. Therefore, the user could manually provide topology data, and such data as component names, and even zone locations. Then the data fed to NTAV3D in real-time would instead be such information as attack segments and compromised components, instead of all data about the network.

b. Inputting Data Into NTAV3D

Either manually or through outside analysis tools, once this data is obtained, it can be converted into an XML file that the thesis application can load, parse, and visualize.

For ease of processing, and to ensure the integrity of data, keeping outputs and inputs in XML format throughout the process would be beneficial. Wireshark has the built-in ability to export its analysis into an XML file. This can be seen in Figure 27 on the following page.

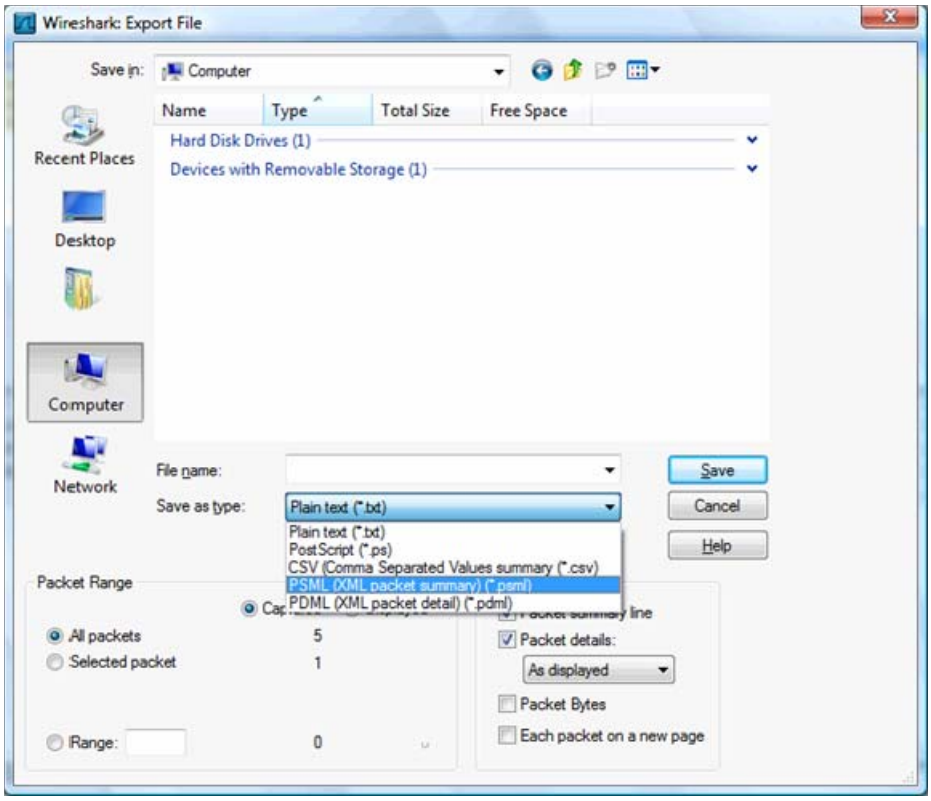


Figure 27. The Export File dialog in Wireshark showing the ability to save the file in PSML format

Snort can also output its detections to an XML file via a plug-in, which can be installed from http://www.cert.org/kb/snortxml (accessed May 2007). Once these network tools have outputs that are in XML format, the aforementioned analysis tools could be run to manipulate and provide data to input to NTAV3D. At this point, an XSLT, or Extensible Stylesheet Language Transformation, can be performed on the file. An XSLT is performed by creating a special kind of XML file. This file contains special rules which are processed against an existing XML file to result in a new file being created that matches the XSLT file's criteria. It is this file that can be made to conform to the standards of the XML parser of NTAV3D. Again, as mentioned in the previous section, an outside tool would have to be utilized to help create the data that this XSLT would execute on the fly. This is because XSLTs can translate data syntactically, but not semantically. This means the XSLT can change the format of data, but not understand its

meaning. For example, the XSLT cannot create the topology data where none exists. But, by using the outside analysis tools, their data can be transformed into a format NTAV3D can understand.

Additionally, there would even be a way to make the transition from CyberCIEGE to network tool input seamless to NTAV3D, in other words, no Java code would have to be changed. Once it is created, the XSLT could be performed such that the new file that will be loaded into NTAV3D, based on the protocol analyzer or intrusion detection system, is in the same format as a CyberCIEGE file. From here, the file can be read into the thesis application in the same way as the CyberCIEGE XML file, and the visualization can continue.

D. EVALUATION OF XJ3D

1. Overall Evaluation

Overall, Xj3D was successfully utilized to meet the objectives of the thesis. As a computer graphics toolkit it, for the most part, produced the desired end result. However, while the problem this thesis attempted to solve is not an easy one, at times finding the solution became not just a matter of overcoming the problem, but one of overcoming the tool chosen. Nonetheless, Xj3D's benefits mean it has potential to be a power tool for producing three dimensional computer programs.

2. Positive Areas

a. X3D Knowledge Transfer

Having a pre-existing knowledge of X3D stand-alone file development allowed the authors a slight head start in programming with SAI and Xj3D. Indeed, once the SAI is understood, it is fairly simple to see how the SAI maps to X3D elements. This is a benefit for those who have worked with stand-alone X3D files, as they can "jump in" to SAI development right away.

b. Graphics Abstraction

Another benefit of Xj3D is the layer of abstraction built into it. The Xj3D SAI allowed the author to focus on content, rather than having to have a deep knowledge of the "inner workings" of how the computer hardware produces a three dimensional scene.

c. Extensibility

Since Xj3D is the Java implementation of the X3D standard, it is quite extensible. What this means is that X3D, and thus Xj3D, has the ability to be expanded, customized, and used in almost any way (within context) that a developer wishes. Customized graphics elements can be created and reused at the will of the developer, rather than being forced to use rigid constructs that may exist with no possibility of enhancement. NTAV3D relies on this ability to create whatever graphical element is needed in order to visualize the information the way that it does. Additionally, the level of extensibility inherent to Xj3D will allow the future expansion of NTAV3D in ways the authors may not even have conceived originally.

3. Areas for Improvement

a. Documentation

As noted in Chapter III, Xj3D was chosen over such a package as OpenSceneGraph because the authors were already familiar with X3D file development, and were more comfortable working with Java. However, while it was noted above this familiarity with X3D allowed a fairly quick understanding of the SAI; it did not provide assistance with actually implementing the SAI. For one, a clear, up to date documentation of how to use the Xj3D SAI in purely programmatic Java is non-existent. All documentation on the Xj3D website is almost exactly a year old; (from 2006) however, development of Xj3D has been continual throughout the past year. Thus, the way the Xj3D SAI is implemented has changed along the way. In some cases in slight ways, and in others, enough to change the way a program is created. Indeed, many person-hours of programming were "wasted" on trying to figure out how to implement X3D nodes that are easily created using X3D file editors. This was mainly due to the

lack of documentation, and having to manually investigate the X3D specification for each node that the authors wanted to create. Some sort of up to date code example repository would help to alleviate this issue, and make using the Xj3D SAI much easier for further programmers.

b. Ease of Programming/Debugging

Along these same lines, the structure of Xj3D does not lend itself to "developer-friendly" programming. For one, the Xj3D SAI objects are not strongly typed. An example is that while there is an Xj3D type known as "X3DNode," to actually create a specific X3D node in the scene graph, simple strings are utilized, as in the example below.

```
X3DNode shape = scene.createNode("Shape");
SFNode shape_geometry = (SFNode) (shape.getField("geometry"));
```

In this code, while a type of X3DNode is defined as "shape," the function "createNode" that has the string "Shape" as it parameter is still required to actually implement that X3D shape node in the scene graph. As can be seen above, this same reliance on strings is seen in the way the SAI gets the fields from the just-created shape node (note the "getField" method with the string "geometry" as the parameter). The result of this weakly typed construction is an increased risk for run-time errors. These errors are generally due to mistyping the string needed, or, again due to lack of documentation, not knowing which exact string to pass in the appropriate methods to create or get the appropriate X3D node or field.

c. X3D Node Implementation

Additionally, some aspects of X3D are buggy or not yet supported by the Xj3D implementation. One example is "bindable" nodes. Upon launch of NTAV3D, it is supposed to jump to a viewpoint, which can be thought of as a specific camera angle for viewing the scene. This is done by setting the "bind" value of the viewpoint to "true" within the SAI Java code. However, though the code matches examples and "should" work, it does not perform as expected. Another example is how some ECMAScript nodes are handled. ECMAScript is a scripting language that allows for the automation or

manipulation of objects within an established environment. (ECMA, 1999) ECMAScript is used within X3D (and thus Xj3D) to provide dynamic control, at run-time, of the three dimensional objects created within the scene graph. This played a role in the navigation scheme. In order to allow keyboard mappings, ECMAScript was necessary. However, the Xj3D SAI could not "route" from the X3D node to the script successfully. Thus, the authors had to resort to creating an actual X3D file on the fly based on viewpoint data, and then integrating it within the scene graph. This process of meshing an outside X3D file with an already created scene graph is known as "inlining." This is not optimal, and violated the NTAV3D design goal of manipulating every X3D element purely programmatically, and with no X3D files. And while the X3D file is in fact created programmatically, and manipulated within Java code, it is not the same as actually using the SAI to create the needed nodes; however, the SAI left the authors no choice. Hopefully, the continued development of Xj3D will allow the resolution of these and similar issues.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. RECOMMENDATIONS FOR FUTURE WORK AND THESIS CONCLUSIONS

A. RECOMMENDATIONS FOR FUTURE WORK

This chapter will first outline work for future extensions to NTAV3D, and will discuss conclusions in Part B.

1. Packaging NTAV3D With CyberCIEGE

Currently, NTAV3D is already able to be packaged with CyberCIEGE. In order to accomplish this task, an "x3d" folder is created in the CyberCIEGE root directory. Into this folder is placed the compiled .class files that make up NTAV3D. Additionally, all of the Xj3D DLL and .jar files should be placed within an "Xj3D" folder within this "x3d" folder. Next, the NTAV3D graphics files are copied to the "\CyberCIEGE\game\exec" directory. Once this is accomplished, the game is launched as normal, and the key combination "ctrl-n" is used from within the game to launch NTAV3D.

Since the capability to be installed and run from within the game already exists, the only remaining task that would have to be accomplished to make NTAV3D a fully redistributable part of CyberCIEGE would be to include the aforementioned files within the CyberCIEGE installer. Xj3D is licensed under a combination of GNU GPL, GNU LGPL, and BSD. (Xj3D Licensing information, May 2007) All of these license structures allow, in general, the Xj3D libraries to be packaged and redistributed.

2. NTAV3D in General

a. Code Enhancement

Future work with NTAV3D includes further enhancements to the program. The code can be optimized for both performance and formatting gains, in terms of class structures, class packaging, memory and program flow optimizations etc. Additionally, as each new version of Xj3D is released, some of the problems mentioned in the previous chapter may be fixed. Thus, it would be beneficial to continually test the NTAV3D codebase against each new snapshot release of Xj3D.

One beneficial modification would be to change the code such that a trail is left during an asset attack so that the user is able to see the entire path that was taken without having to remember it. One method for achieving this would be to have the line segments change color as the information passes along the line segment. Another solution would be to add a bread crumb trail using hash marks behind the information as is travels along.

b. Platform Testing

Additionally, the open source, multi-platform nature of NTAV3D could be employed. Due to hardware available and development constraints, NTAV3D remains untested on Apple OS X or Linux computers. Further testing could be done on these platforms to ensure interoperability.

c. Portability

Since NTAV3D is Java-based, it could conceivably be made into a Java applet, and distributed via the web within web browsers, thus making network visualization quite portable, and perhaps enabling concepts like remote monitoring of security in a three dimensional view from anywhere in the world with internet access.

3. User Studies

Time did not permit user studies to be carried out. Therefore, it would be interesting to see future work carried out that would explore if the network visualizations of NTAV3D actually have a measurable benefit to the user. Such studies would also help to answer the more general question of whether a three dimensional network visualization really does add value to the user experience.

4. Network Tools

Utilizing the framework outlined in the previous chapter, work should be done to allow NTAV3D to work, perhaps in real-time, with network protocol analyzers and intrusion detection systems.

B. CONCLUSIONS

1. Overall Analysis

This thesis has been successful in accomplishing its research motivations and objectives. While time did not permit detailed user studies and analysis, the authors feel that the thesis' network visualization application provides enhancement and benefit to users of CyberCIEGE. Also, a viable framework was presented that will allow those working with network tools to visualize networks, and associated attacks. Finally, as far as can be determined, NTAV3D is the first network visualization software application developed utilizing the combination of XML and Xj3D technology. Thus, the resulting development process allowed for the efficacy of Xj3D as a means for network attack visualization to be successfully evaluated.

The problem of network visualization is not an easy one to resolve, and it is hoped the research presented within this thesis will assist other researchers in the form of recommended practices and lessons learned. Network administrators will *never* have all of the information needed to prevent network intrusion. However, this thesis never had the goal of improving the methods of network intrusion detection. Instead, it *was* successful in being able to provide a better *context* for seeing the information administrators *do* have, and might not have been able to interpret as well without a visual context.

2. Enhancement to CyberCIEGE

The creation of the thesis' NTAV3D application has provided an enhancement to the gameplay of CyberCIEGE that did not exist previously. Prior to the creation of NTAV3D, CyberCIEGE users were only able to view the networks they had constructed in a two dimensional view, and no player interaction was possible, other than changing network connections. The player was merely presented with a basic network map. This view provided no relation to the "office view" of how components were laid out. Additionally, due to the nature of the diagram, it was hard to decipher which networks components with multiple connections were part of. On the contrary, NTAV3D allows the gameplayer the ability to explore the network they have created; to "walk through" the network's configuration in a way similar to the physical office during regular

gameplay. Figures 28 and 29 show this difference. Figure 28 is the current state of how the gameplayer sees the network he or she has created in CyberCIEGE. Conversely, Figure 29 shows the exact same network visualized within NTAV3D. While it is hard to capture the experience of "walking" a three dimensional space in prose, the difference seen in these figures, which is the ability to virtually move through the network in a three dimensional space provides the user with a higher level of interaction than the current "flat," two dimensional view. Additionally, since the locations of the network devices map to their locations in the "office view" of CyberCIEGE, the player should be able to more easily deduce specifically which network component from the game he or she is looking at.

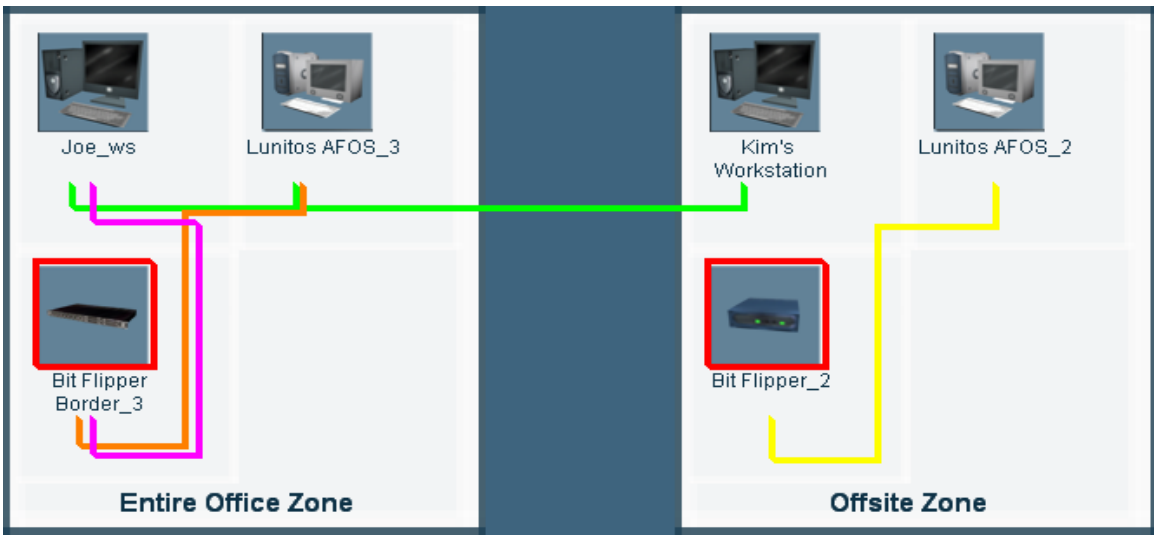


Figure 28. View of CyberCIEGE's current network view

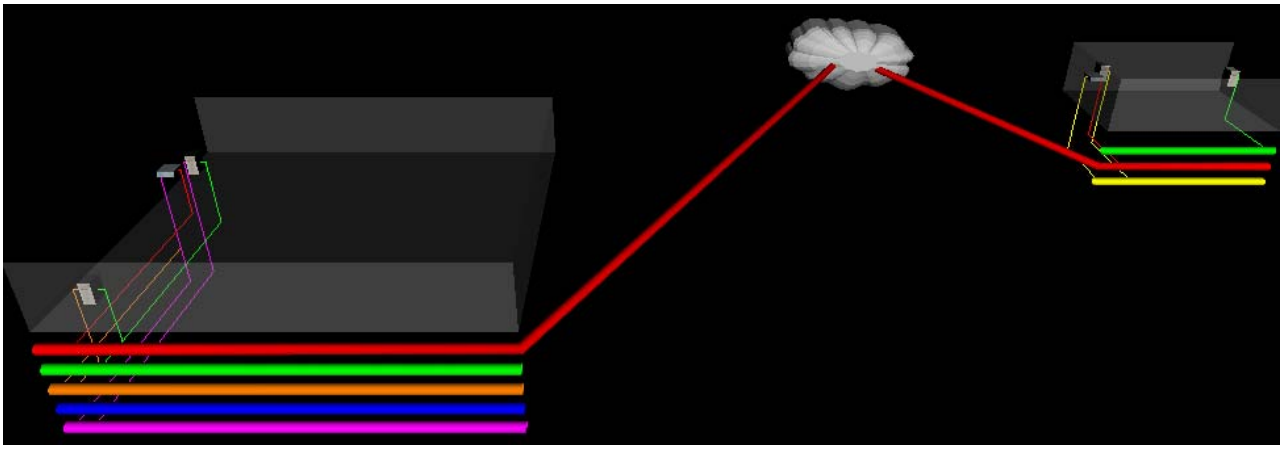


Figure 29. A slightly overhead angle of NTAV3D's view of the same network

Besides just providing a much improved view of the network topology, NTAV3D provides the player with a vastly improved depiction of network attacks. Instead of only the generic movie that would play informing the player of an attack, NTAV3D allows the player to actually see the information flow resulting from an attack. Depicting this in the three dimensional space also allows the player the ability to adjust their view of the attack so that they can glean the best understanding of the security issues being taught. This can be an invaluable training tool in learning about network security, and will be explained more in the section below.

3. Visualizing Network Attacks

a. In CyberCIEGE

Given the information output by CyberCIEGE, users are able to observe the path taken by information during asset attacks. By observing the flow of traffic during one of these attacks, the user is able to determine which type of attack (integrity or secrecy) is occurring. By observing component compromises, players can hypothesize the role of flaws, malicious software, and physical attacks in the attack on the asset. Similarly, by observing malicious software on components, the player can deduce the potential for availability attacks.

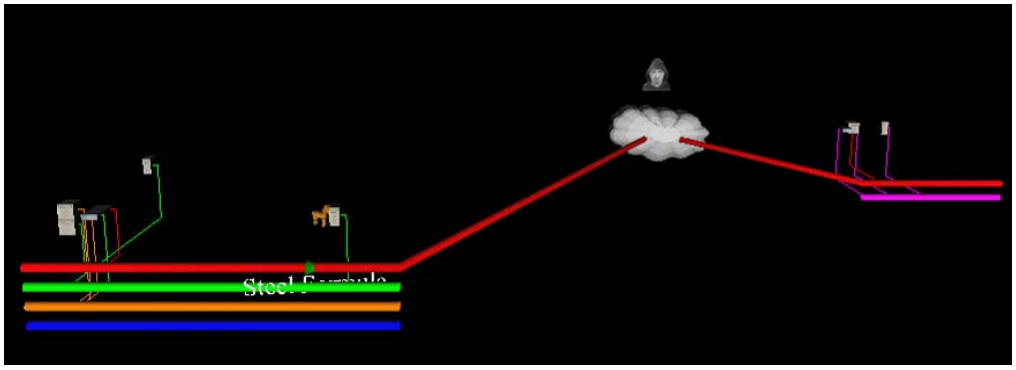


Figure 30. Overhead view of entire scene during an example asset attack with the walls turned off

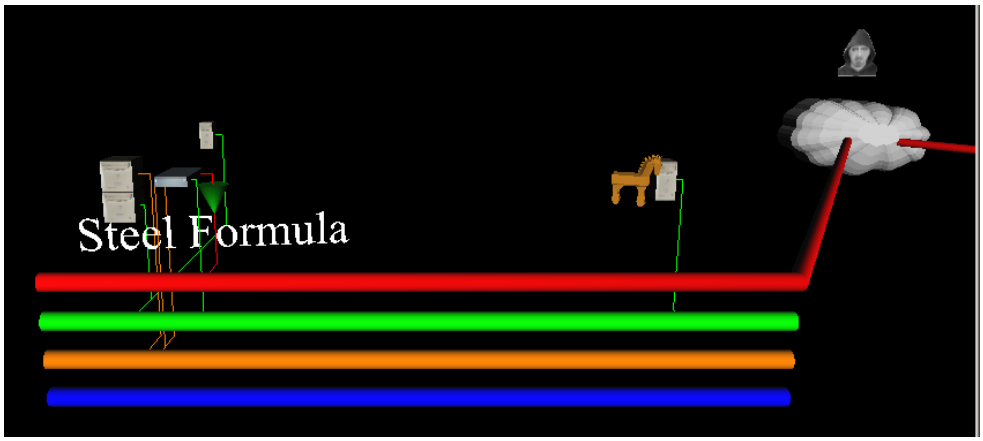


Figure 31. A close up view of the main office during an example asset attack with the walls turned off

Figure 30 and Figure 31 are two different views during an example secrecy asset attack. In this example, there is a Trojan horse on a component and the attacker is out on the web somewhere. The name of the asset is shown because this is a secrecy attack. Because this is just a snap shot, the path taken by the information is not visible, but the direction of travel is known because of the orientation of the cone.

b. In Network Tools

Once the framework in the previous chapter is implemented, NTAV3D has the possibility of being expanded to provide visualization of network attacks detected from such programs as Snort. This could become an excellent tool for network security professionals, as it would provide them with the increased network awareness to help thwart said attack. If network security experts can "see" an attack as its happening, they will have a better chance of isolating and stopping it, and preventing similar attacks in the future.

4. Working With Network Tools

As just stated, the framework provided in the previous chapter could provide NTAV3D with the means to become an excellent tool for network security professionals. The framework could be a viable method for easily utilizing the capabilities of NTAV3D to interact with network protocol analyzers, or intrusion detection systems. By

harnessing the power of XML, and the pre-existing XML parsing in NTAV3D, the software application can successfully add increased network awareness to those charged with keeping it secure.

5. Final Thoughts

By combining research in network topology and vulnerability visualization, three dimensional graphics, and open source programming technologies, this thesis was successfully able to provide valuable enhancement to CyberCIEGE. Additionally, with the future work outlined previously, this NTAV3D project can be extended to provide additional functionality to network security tools. NTAV3D is also one of the first programs to utilize solely the Java-based Xj3D SAI to create three dimensional scenes. Thus, this thesis has provided contributions to three dimensional graphics programming, and the overall goals of improving network security and defense.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. X3D NODE TYPES

A. LIST OF X3D NODES UTILIZED IN NTAV3D

The NTAV3D solution utilizes X3D graphics in only pure Java code, without manipulating an actual X3D format file. This approach may be a less familiar concept to those X3D or VRML developers who usually only work with standard stand-alone X3D or VRML files. Therefore, for reference, the following is a list of the X3D nodes that were implemented within NTAV3D. To understand how each node was implemented, refer to code snippets included throughout this document, the selected classes in Appendix B, or the NTAV3D source, which is available for review. For more information on these nodes, refer to the table derived from the X3D specification found at *http://www.realism.com/Web3D/x3d/nodeReference.html (last accessed, April 30, 2007).*

X3D nodes implemented within NTAV3D listed alphabetically:

- Appearance
- Billboard
- BooleanSequencer
- BooleanTimeTrigger
- Box
- Color
- Cone
- Coordinate
- Cylinder
- GeoViewpoint
- Group
- ImageTexture
- IndexedFaceSet
- Inline
- IntegerSequencer
- IntegerTimeTrigger
- KeySensor
- Material
- MovieTexture
- NavigationInfo
- PositionInterpolator
- ProximitySensor
- Script
- Shape
- Sphere
- Switch
- Text
- TextureCoordinate
- TimeSensor
- TouchSensor
- Transform
- Viewpoint
- WorldInfo

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. SELECTED XJ3D JAVA CODE

A. PARSEXML CLASS (EXCERPT)

```
package utility;

// Core Java APIs
import geoNodes.AttackerIcon;
import geoNodes.Cloud;
import geoNodes.NodeStorage;
import geoNodes.WallCreator;

import java.io.File;
import java.io.IOException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.web3d.x3d.sai.MFString;
import org.web3d.x3d.sai.X3DNode;
import org.web3d.x3d.sai.X3DScene;
import org.xml.sax.SAXException;

import com.sun.org.apache.xerces.internal.parsers.DOMParser;

import dataNodes.AttackStorage;
import dataNodes.CompromiseStorage;
import dataNodes.Network;
import dataNodes.NetworkStorage;

public class ParseXML {

    public ParseXML(X3DScene scene, File parseFile) {
        this.scene = scene;
        this.parseFile = parseFile;
    }

    private X3DScene scene;

    File parseFile;

    private static int hold = 0;

    private static int holdnet = 0;

    private NodeStorage nodeStorage = new NodeStorage();

    private CompromiseStorage compromiseStorage = new CompromiseStorage();

    private AttackStorage attackStorage = new AttackStorage();

    private NetworkStorage networkStorage = new NetworkStorage();

    private WallCreator wallCreator[] = new WallCreator[1];

    private float maxX = 0;

    private float minX = 0;

    private float minZ = 0;

    private float maxZ = 0;
```

```
private boolean internetExist = false;

private boolean leasedExist = false;

private boolean remoteInterExist = false;

private boolean remoteLeasedExist = false;

private int internetEl = -1;

private int leasedEl = -1;

private float[] mainSite;

private float[] remoteSite;

private Network remoteInternet;

private Network remoteLeased;

private float[] internetLinep;

private float[] leasedLinep;

private float[] mainInternetEnd = new float[3];

private float[] mainLeasedEnd = new float[3];

private float[] remoteInternetStart = new float[3];

private float[] remoteLeasedStart = new float[3];

public void parse() {

    try {

        // Create a DOM Parser
        DOMParser parser = new DOMParser();

        // Parse the incoming file (passed from Main)
        parser.parse(parseFile.toURI().toString());

        // Obtain the document
        Document doc = parser.getDocument();

        parseDocument(doc);

    } catch (IOException ioe) {

        ioe.printStackTrace();

    } catch (SAXException saxe) {

        saxe.printStackTrace();
    }
}

private void parseDocument(Document doc) {
    {
        minX = findMinX(doc);
        maxX = findMaxX(doc);

        minZ = findMinZ(doc);
        maxZ = findMaxZ(doc);

        mainSite = findSite(doc, 0);

        remoteSite = findSite(doc, 999);
```

```
new MovingViewpoint(mainSite, remoteSite);

// float z = (minZ + maxZ) / 2;

float z = maxZ;

float y = -2;

float y2 = -2.5f;

remoteInternet = new Network(scene);

remoteLeased = new Network(scene);

NodeList netNodeLst = doc.getElementsByTagName("network");

// adding main pipes
for (int x = 0; x < netNodeLst.getLength(); x++) {
    networkStorage.setNetworkArr(holdnet, scene);

    Node netFstNode = netNodeLst.item(x);

    if (netFstNode.getNodeType() == Node.ELEMENT_NODE) {

        Element netNameElmnt = (Element) netFstNode;
        NodeList netNmElmntLst = netNameElmnt
                .getElementsByTagName("networkName");
        Element netNmElmnt = (Element) netNmElmntLst.item(0);
        NodeList netName = netNmElmnt.getChildNodes();
        networkStorage.getNetworkArrEl(holdnet).setNetworkName(
                (netName.item(0)).getNodeValue());

        NodeList netColElmntLst = netNameElmnt
                .getElementsByTagName("color");
        Element netColElmnt = (Element) netColElmntLst.item(0);
        NodeList netColor = netColElmnt.getChildNodes();
        networkStorage.getNetworkArrEl(holdnet).setColor(
                (netColor.item(0)).getNodeValue());

        if (networkStorage.getNetworkArrEl(holdnet)
        .getNetworkName().toLowerCase()
        .contains("internet")) {
            internetEl = holdnet;
            internetExist = true;
            networkStorage.getNetworkArrEl(holdnet).setLinep(
                    new float[] { mainSite[0], y, z, mainSite[1],
                    y, z });
            mainInternetEnd[0] = mainSite[1];
            mainInternetEnd[1] = y;
            mainInternetEnd[2] = z;
            internetLinep = networkStorage.getNetworkArrEl(holdnet)
            .getLinep();
            // z = z + 2;
        } else if (networkStorage.getNetworkArrEl(holdnet)
        .getNetworkName().toLowerCase().contains("remote")
        || networkStorage.getNetworkArrEl(holdnet)
        .getNetworkName().toLowerCase().contains(
                "offsite")) {
            if ((internetExist == true)
            && (remoteInterExist == false)) {
                remoteInternet.setNetworkName(networkStorage
                        .getNetworkArrEl(internetEl)
                        .getNetworkName());
                remoteInternet.setLinep(new float[] {
                    remoteSite[0], y2, remoteSite[2],
                    remoteSite[1], y2, remoteSite[2] });
```

```
        remoteInternet.setColor(networkStorage
                .getNetworkArrEl(internetEl).getStrColor());
        remoteInternet
                .addInternet(new float[] {
            ((minX + maxX) / 2 + 10), -2,
            (minZ - 10) });
        remoteInterExist = true;
        remoteInternetStart[0] = remoteSite[0];
        remoteInternetStart[1] = y2;
        remoteInternetStart[2] = remoteSite[2];
        // remoteSite[2] = remoteSite[2] + 2;
        y2 = y2 - 1.25f;
    }
    networkStorage.getNetworkArrEl(holdnet).setLinep(
            new float[] { remoteSite[0], y2, remoteSite[2],
            remoteSite[1], y2, remoteSite[2] });
    // remoteSite[2] = remoteSite[2] + 2;
    y2 = y2 - 1.25f;
} else {
    networkStorage.getNetworkArrEl(holdnet).setLinep(
            new float[] { mainSite[0], y, z, mainSite[1],
            y, z });
    // z = z + 2;
}

if (networkStorage.getNetworkArrEl(holdnet)
.getNetworkName().toLowerCase().contains("leased")) {
    leasedEl = holdnet;
    leasedExist = true;
    networkStorage.getNetworkArrEl(holdnet).setLinep(
            new float[] { mainSite[0], y, z, mainSite[1],
            y, z });
    mainLeasedEnd[0] = mainSite[1];
    mainLeasedEnd[1] = y;
    mainLeasedEnd[2] = z;
    leasedLinep = networkStorage.getNetworkArrEl(holdnet)
    .getLinep();
    // z = z + 2;
} else if ((leasedExist == true)
&& (remoteLeasedExist == false)) {
    remoteLeased.setNetworkName(networkStorage
            .getNetworkArrEl(leasedEl).getNetworkName());
    remoteLeased
            .setLinep(new float[] { remoteSite[0], y2,
            remoteSite[2], remoteSite[1], y2,
            remoteSite[2] });
    remoteLeased.setColor(networkStorage.getNetworkArrEl(
            leasedEl).getStrColor());
    remoteLeased.addNode();
    remoteLeasedExist = true;
    remoteLeasedStart[0] = remoteSite[0];
    remoteLeasedStart[1] = y2;
    remoteLeasedStart[2] = remoteSite[2];
    // remoteSite[2] =
    // remoteSite[2] + 2;
    y2 = y2 - 1.25f;
}

if (holdnet == internetEl) {
    networkStorage.getNetworkArrEl(x).addInternet(
            new float[] { ((minX + maxX) / 2 + 10), -2,
            (minZ - 10) });
} else {
    networkStorage.getNetworkArrEl(holdnet).addNode();
}

y = y - 1.25f;
holdnet++;
```

```
        }
    }

    networkStorage.setNetworkArr(holdnet, scene);
    networkStorage.getNetworkArrEl(holdnet).setColor("0x00000000");
    networkStorage.getNetworkArrEl(holdnet).setNetworkName(
            "NullNet7834");
    networkStorage.getNetworkArrEl(holdnet).setLinep(
            new float[] { 0, 0, 0, 0, 0, 0 });
}

{
    NodeList nodeLst = doc.getElementsByTagName("component");

    for (int x = 0; x < nodeLst.getLength(); x++) {
        nodeStorage.setBoxArr(hold, scene, new float[] { 0, 0, 0 },
                new float[] { 0, 0, 0, 0 }, new float[] { 0.5f, 0.5f,
                0.5f }, new float[] { 0.8f, 0.8f, 0.7f });

        Node fstNode = nodeLst.item(x);

        if (fstNode.getNodeType() == Node.ELEMENT_NODE) {

            Element comNameElmnt = (Element) fstNode;
            NodeList comNmElmntLst = comNameElmnt
                    .getElementsByTagName("componentName");
            Element comNmElmnt = (Element) comNmElmntLst.item(0);
            NodeList comName = comNmElmnt.getChildNodes();
            nodeStorage.getBoxArrEl(hold).setCompName(
                    (comName.item(0)).getNodeValue());

            NodeList comBaseNmElmntLst = comNameElmnt
                    .getElementsByTagName("componentBase");
            Element comBaseNmElmnt = (Element) comBaseNmElmntLst
                    .item(0);
            NodeList comBaseName = comBaseNmElmnt.getChildNodes();
            nodeStorage.getBoxArrEl(hold).setCompBase(
                    (comBaseName.item(0)).getNodeValue());

            NodeList assetNmElmntLst = comNameElmnt
                    .getElementsByTagName("assetName");
            Element assetNmElmnt = (Element) assetNmElmntLst.item(0);
            try {

                NodeList assetName = assetNmElmnt.getChildNodes();
                nodeStorage.getBoxArrEl(hold).setAsset(
                        (assetName.item(0)).getNodeValue());
            } catch (NullPointerException e) {
                nodeStorage.getBoxArrEl(hold).setAsset("");
            }

            if (nodeStorage.getBoxArrEl(hold).getCompBase()
            .toLowerCase().contains("router")
            || nodeStorage.getBoxArrEl(hold).getCompBase()
            .toLowerCase().contains("flipper")
            || nodeStorage.getBoxArrEl(hold).getCompBase()
            .toLowerCase().contains("asbestos")
            || nodeStorage.getBoxArrEl(hold).getCompBase()
            .toLowerCase().contains("linkcrypt")) {
                nodeStorage.getBoxArrEl(hold).setCompType(2);
                nodeStorage.getBoxArrEl(hold).setScal(
                        new float[] { 0.5f, 0.15f, 0.5f });
            } else if (nodeStorage.getBoxArrEl(hold).getCompBase()
            .toLowerCase().contains("desktop")
            || nodeStorage.getBoxArrEl(hold).getCompBase()
            .toLowerCase().contains("green")
            || nodeStorage.getBoxArrEl(hold).getCompBase()
```

```
                    .toLowerCase().contains("lunitos")) {
                        nodeStorage.getBoxArrEl(hold).setCompType(3);
                        nodeStorage.getBoxArrEl(hold).setScal(
                                new float[] { 0.3f, 0.6f, 0.4f });
                    } else if (nodeStorage.getBoxArrEl(hold).getCompBase()
                    .toLowerCase().contains("server")) {
                        nodeStorage.getBoxArrEl(hold).setCompType(1);
                    }

                    NodeList netNmElmntLst = comNameElmnt
                            .getElementsByTagName("networkName");
                    for (int y = 0; y < netNmElmntLst.getLength(); y++) {
                        Element netNmElmnt = (Element) netNmElmntLst.item(y);
                        NodeList netName = netNmElmnt.getChildNodes();
                        nodeStorage.getBoxArrEl(hold).setNetworks(y,
                                (netName.item(0)).getNodeValue());
                    }
. . .
}
```

B. WALLCREATOR CLASS

```
package geoNodes;

import org.web3d.x3d.sai.MFInt32;
import org.web3d.x3d.sai.MFNode;
import org.web3d.x3d.sai.MFString;
import org.web3d.x3d.sai.MFVec3f;
import org.web3d.x3d.sai.SFBool;
import org.web3d.x3d.sai.SFColor;
import org.web3d.x3d.sai.SFFloat;
import org.web3d.x3d.sai.SFInt32;
import org.web3d.x3d.sai.SFNode;
import org.web3d.x3d.sai.SFVec3f;
import org.web3d.x3d.sai.X3DNode;
import org.web3d.x3d.sai.X3DScene;

public class WallCreator {

    private X3DScene scene;

    private float[] corner1;

    private float[] corner2;

    private float[] col;

    private float transparency;

    private String[] zoneName = { "" };

    public WallCreator() {
    }

    public WallCreator(X3DScene scene, float[] input1, float[] input2) {
        this.scene = scene;
        this.corner1 = input1;
        this.corner2 = input2;
        float[] color = { .8f, .8f, .8f };
```

```
        this.col = color;
        float transp = .95f;
        this.transparency = transp;
    }

    public void setZoneName(String name) {
        this.zoneName[0] = name;
    }

    public String getZoneName() {
        return this.zoneName[0];
    }

    public float[] getXCoords() {
        return new float[] { corner1[0], corner2[0] };
    }

    public void addNode() {

        X3DNode switchNode = (X3DNode) scene.createNode("Switch");
        MFNode switch_children = (MFNode) switchNode.getField("children");
        switch_children.clear();
        SFInt32 whichChoice = (SFInt32) switchNode.getField("whichChoice");
        whichChoice.setValue(0);

        X3DNode tr = (X3DNode) scene.createNode("Transform");
        MFNode tran_children = (MFNode) tr.getField("children");
        tran_children.clear();

        X3DNode shape = scene.createNode("Shape");
        SFNode shape_geometry = (SFNode) (shape.getField("geometry"));
        X3DNode box = scene.createNode("IndexedFaceSet");
        SFBool solid = (SFBool) box.getField("solid");
        solid.setValue(false);
        SFBool convex = (SFBool) box.getField("convex");
        convex.setValue(false);
        X3DNode appearance = scene.createNode("Appearance");
        SFNode shape_appearance = (SFNode) (shape.getField("appearance"));
        SFNode appear_material = (SFNode) (appearance.getField("material"));
        X3DNode material = scene.createNode("Material");
        SFColor mat_color = (SFColor) (material.getField("diffuseColor"));
        mat_color.setValue(col);
        SFFloat mat_transparency = (SFFloat) (material.getField("transparency"));
        mat_transparency.setValue(transparency);
        appear_material.setValue(material);
        MFInt32 coord_index = (MFInt32) (box.getField("coordIndex"));
        coord_index.setValue(30, new int[] { 4, 5, 1, 0, 4, -1, 5, 6, 2, 1, 5, -1, 6, 7, 3,
2, 6, -1, 7, 4, 0, 3, 7, -1, 0, 3, 2, 1, 0, -1, });
        SFNode line_coord = (SFNode) (box.getField("coord"));
        X3DNode coordinate = scene.createNode("Coordinate");
        MFVec3f point_value = (MFVec3f) (coordinate.getField("point"));
        point_value.setValue(8, new float[] { this.corner1[0], -1,
        this.corner1[1], // 0
        this.corner2[0], -1, this.corner1[1], // 1
        this.corner2[0], -1, this.corner2[1], // 2
        this.corner1[0], -1, this.corner2[1], // 3
        this.corner1[0], 3, this.corner1[1], // 4
        this.corner2[0], 3, this.corner1[1], // 5
        this.corner2[0], 3, this.corner2[1], // 6
        this.corner1[0], 3, this.corner2[1], // 7
        });
        line_coord.setValue(coordinate);

        // TouchSensor ****Important*****
        X3DNode touchSensor = (X3DNode) scene.createNode("TouchSensor");
        scene.addRootNode(touchSensor);

        shape_appearance.setValue(appearance);
        shape_geometry.setValue(box);
```

```
tran_children.append(shape);
tran_children.append(touchSensor);
switch_children.append(tr);
scene.addRootNode(switchNode);

// KeySensor
X3DNode keySensor = (X3DNode) scene.createNode("KeySensor");
scene.addRootNode(keySensor);

// BooleanFilter1
X3DNode bf1 = (X3DNode) scene.createNode("BooleanFilter");
scene.addRootNode(bf1);

// BooleanFilter2
X3DNode bf2 = (X3DNode) scene.createNode("BooleanFilter");
scene.addRootNode(bf2);

// BooleanFilter3
X3DNode bf3 = (X3DNode) scene.createNode("BooleanFilter");
scene.addRootNode(bf3);

// IntegerTriggers for turning walls on/off
X3DNode triggerOn = (X3DNode) scene.createNode("IntegerTrigger");
SFInt32 integerKeyOn = (SFInt32) triggerOn.getField("integerKey");
integerKeyOn.setValue(0);
scene.addRootNode(triggerOn);

X3DNode triggerOff = (X3DNode) scene.createNode("IntegerTrigger");
SFInt32 integerKeyOff = (SFInt32) triggerOff.getField("integerKey");
integerKeyOff.setValue(-1);
scene.addRootNode(triggerOff);

// IntegerTriggers for mouseover text
X3DNode triggerTextOn = (X3DNode) scene.createNode("IntegerTrigger");
SFInt32 integerKeyOnText = (SFInt32) triggerTextOn
        .getField("integerKey");
integerKeyOnText.setValue(0);
scene.addRootNode(triggerTextOn);

X3DNode triggerTextOff = (X3DNode) scene.createNode("IntegerTrigger");
SFInt32 integerKeyOffText = (SFInt32) triggerTextOff
        .getField("integerKey");
integerKeyOffText.setValue(-1);
scene.addRootNode(triggerTextOff);

// Switch Node for mouse-over text
X3DNode switchNodeText = (X3DNode) scene.createNode("Switch");
MFNode switchText_children = (MFNode) switchNodeText
        .getField("children");
switchText_children.clear();
SFInt32 whichChoiceText = (SFInt32) switchNodeText
        .getField("whichChoice");
whichChoiceText.setValue(-1);

// Billboard for text
X3DNode bb = (X3DNode) scene.createNode("Billboard");
SFVec3f rotationAxis = (SFVec3f) bb.getField("axisOfRotation");
rotationAxis.setValue(new float[] { 0, 1, 0 });
MFNode bill_children = (MFNode) bb.getField("children");
bill_children.clear();

// Transform to locate text
X3DNode tr2 = (X3DNode) scene.createNode("Transform");
SFVec3f translation2 = (SFVec3f) tr2.getField("translation");
translation2.setValue(new float[] {
    (this.corner1[0] + this.corner2[0]) / 2.0f, 1.5f,
    (this.corner1[1] + 1.5f) });
SFVec3f scale2 = (SFVec3f) tr2.getField("scale");
scale2.setValue(new float[] { 2.5f, 2.5f, .5f });
MFNode tran_children2 = (MFNode) tr2.getField("children");
tran_children2.clear();
```

```
        // Create text shape
        X3DNode shape2 = scene.createNode("Shape");
        SFNode shape_geometry2 = (SFNode) (shape2.getField("geometry"));
        X3DNode text = scene.createNode("Text");
        MFString words = (MFString) (text.getField("string"));
        words.setValue(zoneName.length, zoneName);

        // Justify text for middle middle
        X3DNode fontStyle = (X3DNode) scene.createNode("FontStyle");
        SFNode text_justify = (SFNode) text.getField("fontStyle");
        MFString justify = (MFString) fontStyle.getField("justify");
        justify.setValue(2, new String[] { "MIDDLE", "MIDDLE" });
        text_justify.setValue(fontStyle);

        X3DNode appearance2 = scene.createNode("Appearance");
        SFNode shape_appearance2 = (SFNode) (shape2.getField("appearance"));
        SFNode appear_material2 = (SFNode) (appearance2.getField("material"));
        X3DNode material2 = scene.createNode("Material");
        SFColor mat_color2 = (SFColor) (material2.getField("diffuseColor"));
        mat_color2.setValue(new float[] { 1, 1, 1 });
        appear_material2.setValue(material2);
        shape_appearance2.setValue(appearance2);
        shape_geometry2.setValue(text);
        bill_children.append(shape2);
        tran_children2.append(bb);
        switchText_children.append(tr2);
        scene.addRootNode(switchNodeText);

        // Routes
        scene.addRoute(keySensor, "altKey", bf1, "set_boolean");
        scene.addRoute(keySensor, "controlKey", bf2, "set_boolean");
        scene.addRoute(bf1, "inputTrue", triggerOff, "set_boolean");
        scene.addRoute(bf2, "inputTrue", triggerOn, "set_boolean");
        scene.addRoute(triggerOff, "triggerValue", switchNode,
                "set_whichChoice");
        scene
                .addRoute(triggerOn, "triggerValue", switchNode,
                "set_whichChoice");
        scene.addRoute(touchSensor, "isOver", bf3, "set_boolean");
        scene.addRoute(bf3, "inputTrue", triggerTextOn, "set_boolean");
        scene.addRoute(bf3, "inputFalse", triggerTextOff, "set_boolean");
        scene.addRoute(triggerTextOn, "triggerValue", switchNodeText,
                "set_whichChoice");
        scene.addRoute(triggerTextOff, "triggerValue", switchNodeText,
                "set_whichChoice");

    }

}
```

C. ATTACKERICON CLASS

```
package geoNodes;

import org.web3d.x3d.sai.MFInt32;
import org.web3d.x3d.sai.MFNode;
import org.web3d.x3d.sai.MFString;
import org.web3d.x3d.sai.MFVec2f;
import org.web3d.x3d.sai.MFVec3f;
import org.web3d.x3d.sai.SFBool;
import org.web3d.x3d.sai.SFColor;
import org.web3d.x3d.sai.SFFloat;
import org.web3d.x3d.sai.SFInt32;
import org.web3d.x3d.sai.SFNode;
import org.web3d.x3d.sai.SFRotation;
import org.web3d.x3d.sai.SFVec3f;
import org.web3d.x3d.sai.X3DNode;
```

```
import org.web3d.x3d.sai.X3DScene;

public class AttackerIcon {

    private X3DScene scene;

    private float[] trans = new float[3];

    private float[] rot = new float[4];

    private float transparency = 0;

    public AttackerIcon() {

    }

    public AttackerIcon(X3DScene scene, float[] position) {
        this.scene = scene;
        this.trans[0] = position[0] - 1.0f;
        this.trans[1] = position[1] - .4f;
        this.trans[2] = position[2];
    }

    public void addNode() {
        X3DNode tr = (X3DNode) scene.createNode("Transform");
        SFVec3f translation = (SFVec3f) tr.getField("translation");
        translation.setValue(trans);
        SFRotation rotation = (SFRotation) tr.getField("rotation");
        rotation.setValue(rot);
        SFVec3f scale = (SFVec3f) tr.getField("scale");
        scale.setValue(new float[] { 1.5f, 1.5f, 1 });
        MFNode tran_children = (MFNode) tr.getField("children");
        tran_children.clear();

        // Attacker shape
        X3DNode shape = scene.createNode("Shape");
        SFNode shape_geometry = (SFNode) (shape.getField("geometry"));
        X3DNode attacker = scene.createNode("IndexedFaceSet");
        SFBool solid = (SFBool) attacker.getField("solid");
        solid.setValue(false);
        X3DNode appearance = scene.createNode("Appearance");
        SFNode shape_appearance = (SFNode) (shape.getField("appearance"));
        SFNode appear_material = (SFNode) (appearance.getField("material"));
        X3DNode material = scene.createNode("Material");
        SFColor mat_color = (SFColor) (material.getField("diffuseColor"));
        mat_color.setValue(new float[] { .2f, .2f, .2f });
        SFFloat mat_transparency = (SFFloat) (material.getField("transparency"));
        mat_transparency.setValue(transparency);
        appear_material.setValue(material);

        SFNode appear_texture = null;
        X3DNode image_texture = null;
        MFString textureURL = null;

        appear_texture = (SFNode) (appearance.getField("texture"));
        image_texture = scene.createNode("ImageTexture");
        textureURL = (MFString) (image_texture.getField("url"));
        textureURL.setValue(1, new String[] { "images.jpg" });
        appear_texture.setValue(image_texture);

        MFInt32 coord_index = (MFInt32) (attacker.getField("coordIndex"));
        MFInt32 texCoord_index = (MFInt32) (attacker.getField("texCoordIndex"));
        coord_index.setValue(coordIndex.length, coordIndex);
        texCoord_index.setValue(textCoordIndex.length, textCoordIndex);
        SFNode line_coord = (SFNode) (attacker.getField("coord"));
        SFNode text_coord = (SFNode) (attacker.getField("texCoord"));
```

```
        X3DNode coordinate = scene.createNode("Coordinate");
        X3DNode textureCoord = scene.createNode("TextureCoordinate");
        MFVec3f point_value = (MFVec3f) (coordinate.getField("point"));
        MFVec2f textCoord_value = (MFVec2f) (textureCoord.getField("point"));
        point_value.setValue(points.length / 3, points);
        textCoord_value.setValue(textureCoordinates.length / 2,
                textureCoordinates);
        line_coord.setValue(coordinate);
        text_coord.setValue(textureCoord);

//              TouchSensor ****Important*****
        X3DNode touchSensor = (X3DNode) scene.createNode("TouchSensor");
        scene.addRootNode(touchSensor);

        shape_appearance.setValue(appearance);
        shape_geometry.setValue(attacker);
        tran_children.append(shape);
        tran_children.append(touchSensor);
        scene.addRootNode(tr);
        scene.updateNamedNode("tr", tr);

//              BooleanFilter
        X3DNode bf = (X3DNode) scene.createNode("BooleanFilter");
        scene.addRootNode(bf);

        // IntegerTriggers
        X3DNode triggerOn = (X3DNode) scene.createNode("IntegerTrigger");
        SFInt32 integerKeyOn = (SFInt32) triggerOn.getField("integerKey");
        integerKeyOn.setValue(0);
        scene.addRootNode(triggerOn);

        X3DNode triggerOff = (X3DNode) scene.createNode("IntegerTrigger");
        SFInt32 integerKeyOff = (SFInt32) triggerOff.getField("integerKey");
        integerKeyOff.setValue(-1);
        scene.addRootNode(triggerOff);

        X3DNode switchNode = (X3DNode) scene.createNode("Switch");
        MFNode switch_children = (MFNode) switchNode.getField("children");
        switch_children.clear();
        SFInt32 whichChoice = (SFInt32) switchNode.getField("whichChoice");
        whichChoice.setValue(-1);

        String[] inputText = { "Attacker" };

        X3DNode bb = (X3DNode) scene.createNode("Billboard");
        SFVec3f rotationAxis = (SFVec3f) bb.getField("axisOfRotation");
        rotationAxis.setValue(new float[] { 0, 1, 0 });
        MFNode bill_children = (MFNode) bb.getField("children");
        bill_children.clear();

        X3DNode tr2 = (X3DNode) scene.createNode("Transform");
        SFVec3f translation2 = (SFVec3f) tr2.getField("translation");
        translation2
                .setValue(new float[] { trans[0], (trans[1]+4.5f), trans[2] });
        SFVec3f scale2 = (SFVec3f) tr2.getField("scale");
        scale2.setValue(new float[] { 2.5f, 2.5f, .5f });
        MFNode tran_children2 = (MFNode) tr2.getField("children");
        tran_children2.clear();

        X3DNode shape2 = scene.createNode("Shape");
        SFNode shape_geometry2 = (SFNode) (shape2.getField("geometry"));
        X3DNode text = scene.createNode("Text");
        MFString words = (MFString) (text.getField("string"));
        words.setValue(inputText.length, inputText);

        X3DNode fontStyle = (X3DNode) scene.createNode("FontStyle");
        SFNode text_justify = (SFNode) text.getField("fontStyle");
```

```
        MFString justify = (MFString) fontStyle.getField("justify");
        justify.setValue(2, new String[] { "MIDDLE", "MIDDLE" });
        text_justify.setValue(fontStyle);

        X3DNode appearance2 = scene.createNode("Appearance");
        SFNode shape_appearance2 = (SFNode) (shape2.getField("appearance"));
        SFNode appear_material2 = (SFNode) (appearance2.getField("material"));
        X3DNode material2 = scene.createNode("Material");
        SFColor mat_color2 = (SFColor) (material2.getField("diffuseColor"));
        mat_color2.setValue(new float[] { 0, 0, 1 });
        appear_material2.setValue(material2);

        shape_appearance2.setValue(appearance2);
        shape_geometry2.setValue(text);
        bill_children.append(shape2);
        tran_children2.append(bb);

        switch_children.append(tr2);
        scene.addRootNode(switchNode);

        scene.addRoute(touchSensor, "isOver", bf, "set_boolean");
        scene.addRoute(bf, "inputTrue", triggerOn, "set_boolean");
        scene.addRoute(bf, "inputFalse", triggerOff, "set_boolean");
        scene
                .addRoute(triggerOn, "triggerValue", switchNode,
                "set_whichChoice");
        scene.addRoute(triggerOff, "triggerValue", switchNode,
                "set_whichChoice");
    }

    float[] points = { 1f, -1.2f, .1f, 1f, -.73594f, .1f, 1f, -.73594f, -.1f, 1f, -1.2f, -
.1f, -1f, -1.2f, .1f, . . ., -.4844f, -.1f };

    int[] coordIndex = { 12, 52, 13, -1, 13, 52, 53, -1, 13, 53, 55, -1, 55, 53, 54, -1,
12, 13, 14, -1, . . ., 68, -1 };

    float[] textureCoordinates = { 1f, 0f, 1f, .193f, 1f, .193f, 1f, 0f, 0f, 0f, 0f, .153f,
0f, .153f, 0f, 0f, .686f, .874f, .314f, .874f,
    .686f, .874f, .314f, .874f, .864f, . . ., .298f };

    int[] textCoordIndex = { 12, 52, 13, -1, 13, 52, 53, -1, 13, 53, 55, -1, 55, 53, 54, -
1, 12, 13, 14, -1, . . ., 68, -1 };

}
```

D. PHYSICAL ATTACK CLASS

```
package geoNodes;

import org.web3d.x3d.sai.MFFloat;
import org.web3d.x3d.sai.MFNode;
import org.web3d.x3d.sai.MFString;
import org.web3d.x3d.sai.MFVec3f;
import org.web3d.x3d.sai.SFBool;
import org.web3d.x3d.sai.SFColor;
import org.web3d.x3d.sai.SFNode;
import org.web3d.x3d.sai.SFTime;
import org.web3d.x3d.sai.SFVec3f;
import org.web3d.x3d.sai.X3DNode;
import org.web3d.x3d.sai.X3DScene;

public class PhysicalAttack {

    private X3DScene scene;
```

```
private float[] trans;

private float[] col;

public PhysicalAttack() {

}

public PhysicalAttack(X3DScene scene, float[] location) {
    this.scene = scene;
    this.trans = location;
    float[] color = { 1, 1, 1 };
    this.col = color;
}

public void addNode() {

    float[] keys = { 0, .25f, .75f, 1 };
    float[] keyValues = { 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1 };

    X3DNode bb = (X3DNode) scene.createNode("Billboard");
    SFVec3f rotationAxis = (SFVec3f) bb.getField("axisOfRotation");
    rotationAxis.setValue(new float[] { 0, 1, 0 });
    MFNode bill_children = (MFNode) bb.getField("children");
    bill_children.clear();

    String[] name = { "Component", "Stolen!" };

    X3DNode tr2 = (X3DNode) scene.createNode("Transform");
    SFVec3f translation2 = (SFVec3f) tr2.getField("translation");
    translation2
            .setValue(new float[] { (trans[0] - 2), trans[1], trans[2] });
    SFVec3f scale2 = (SFVec3f) tr2.getField("scale");
    scale2.setValue(new float[] { 3.0f, 3.0f, 1 });
    MFNode tran_children2 = (MFNode) tr2.getField("children");
    tran_children2.clear();

    X3DNode shape2 = scene.createNode("Shape");
    SFNode shape_geometry2 = (SFNode) (shape2.getField("geometry"));
    X3DNode text = scene.createNode("Text");
    MFString words = (MFString) (text.getField("string"));
    words.setValue(name.length, name);

    X3DNode fontStyle = (X3DNode) scene.createNode("FontStyle");
    SFNode text_justify = (SFNode) text.getField("fontStyle");
    MFString justify = (MFString) fontStyle.getField("justify");
    justify.setValue(2, new String[] { "MIDDLE", "MIDDLE" });
    text_justify.setValue(fontStyle);

    X3DNode appearance2 = scene.createNode("Appearance");
    SFNode shape_appearance2 = (SFNode) (shape2.getField("appearance"));
    SFNode appear_material2 = (SFNode) (appearance2.getField("material"));
    X3DNode material2 = scene.createNode("Material");
    SFColor mat_color2 = (SFColor) (material2.getField("diffuseColor"));
    mat_color2.setValue(new float[] { 1, 1, 1 });
    appear_material2.setValue(material2);

    shape_appearance2.setValue(appearance2);
    shape_geometry2.setValue(text);
    bill_children.append(shape2);
    tran_children2.append(bb);

    scene.addRootNode(tr2);
    scene.updateNamedNode("tr2", tr2);

    // positionInterpolator
```

```
        X3DNode pi = (X3DNode) scene.createNode("PositionInterpolator");
        MFFloat key = (MFFloat) pi.getField("key");
        key.setValue(keys.length, keys);
        MFVec3f keyValue = (MFVec3f) pi.getField("keyValue");
        keyValue.setValue((keyValues.length) / 3, keyValues);
        scene.addRootNode(pi);
        scene.updateNamedNode("pi", pi);

        // TimeSensor
        X3DNode timeSensor = (X3DNode) scene.createNode("TimeSensor");
        SFBool loop = (SFBool) timeSensor.getField("loop");
        loop.setValue(true);
        SFTime cycleInterval = (SFTime) timeSensor.getField("cycleInterval");
        cycleInterval.setValue(2.0);
        scene.addRootNode(timeSensor);
        scene.updateNamedNode("timeSensor", timeSensor);

        // Routes
        scene.addRoute(timeSensor, "fraction_changed", pi, "set_fraction");
        scene.addRoute(pi, "value_changed", material2, "diffuseColor");
    }

}
```

APPENDIX C. THESIS CLASS DIAGRAM

A. THESIS APPLICATION CLASS DIAGRAM

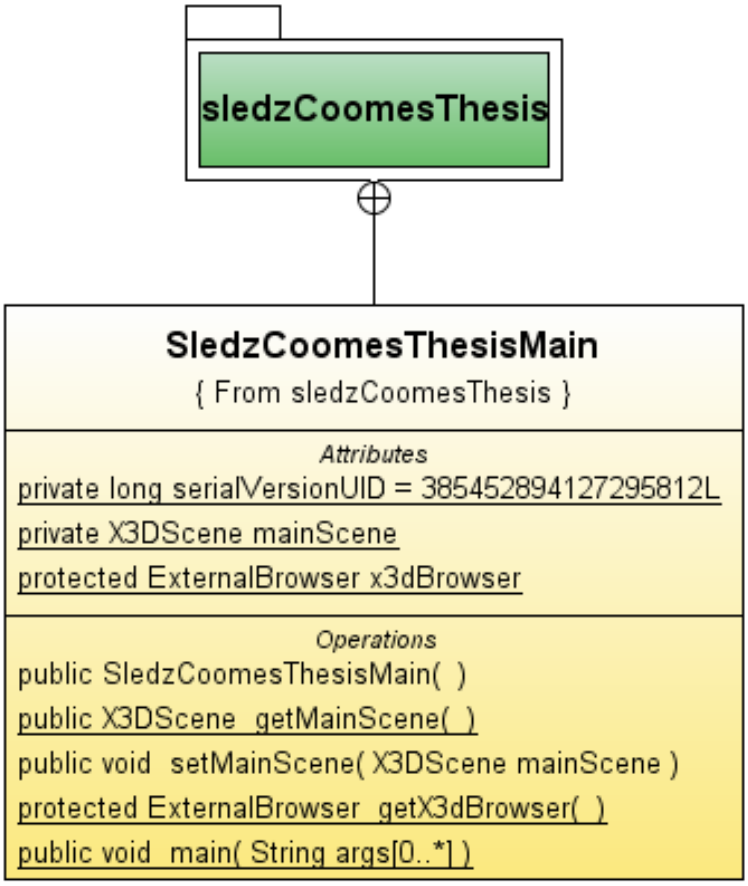
The following pages contain UML class diagrams for each programmatic package of the thesis application. These diagrams allow the classes to be seen at a glace, and to see which classes relate to each other. To organize the programming code, classes with similar functions are grouped into what are known as "packages." Due to space limitations and to enhance readability, in some cases, the packages are broken up into parts for the purposes of these diagrams, but each class in the "part1, 2, etc." diagram is actually all within one package. The following is a brief summary of what each package contains.

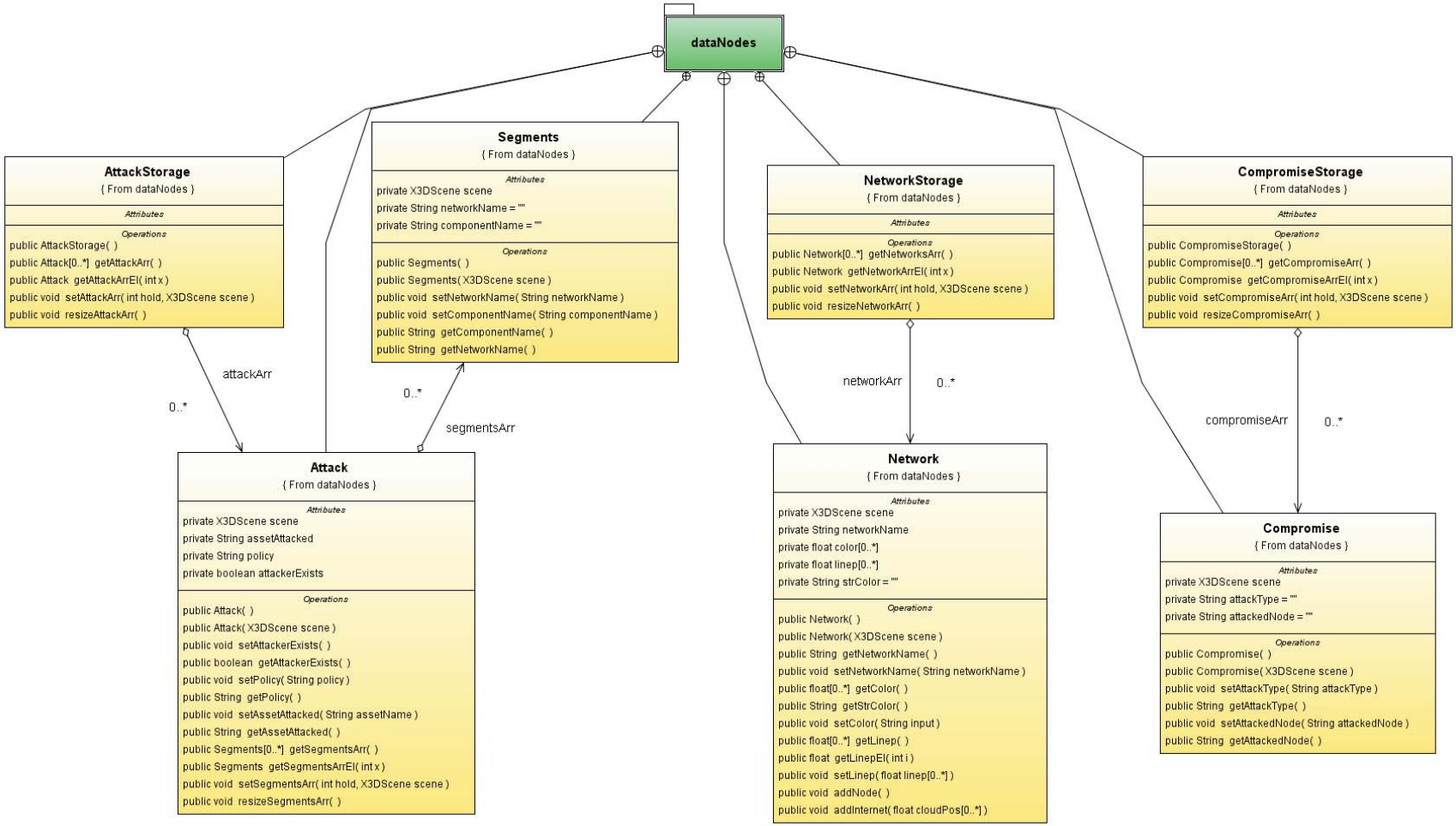
sledzCoomesThesis: Contains the "main" method of the program, as well as those that setup the Xj3D browsing environment. It also contains the calls to the XML parsing class, which starts the process of visualizing whatever is input to NTAV3D.

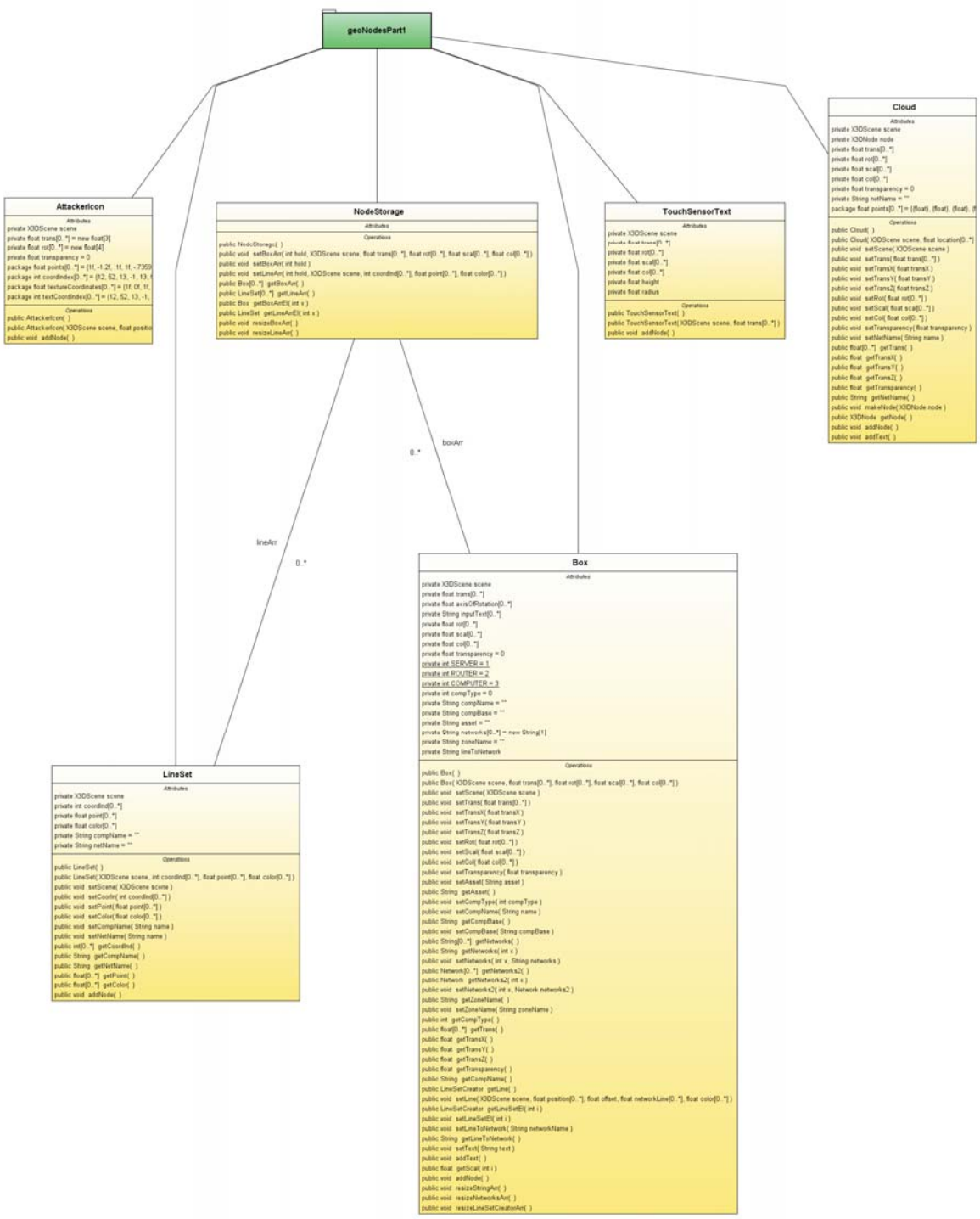
dataNodes: Contain classes that hold data about network nodes and network attack information.

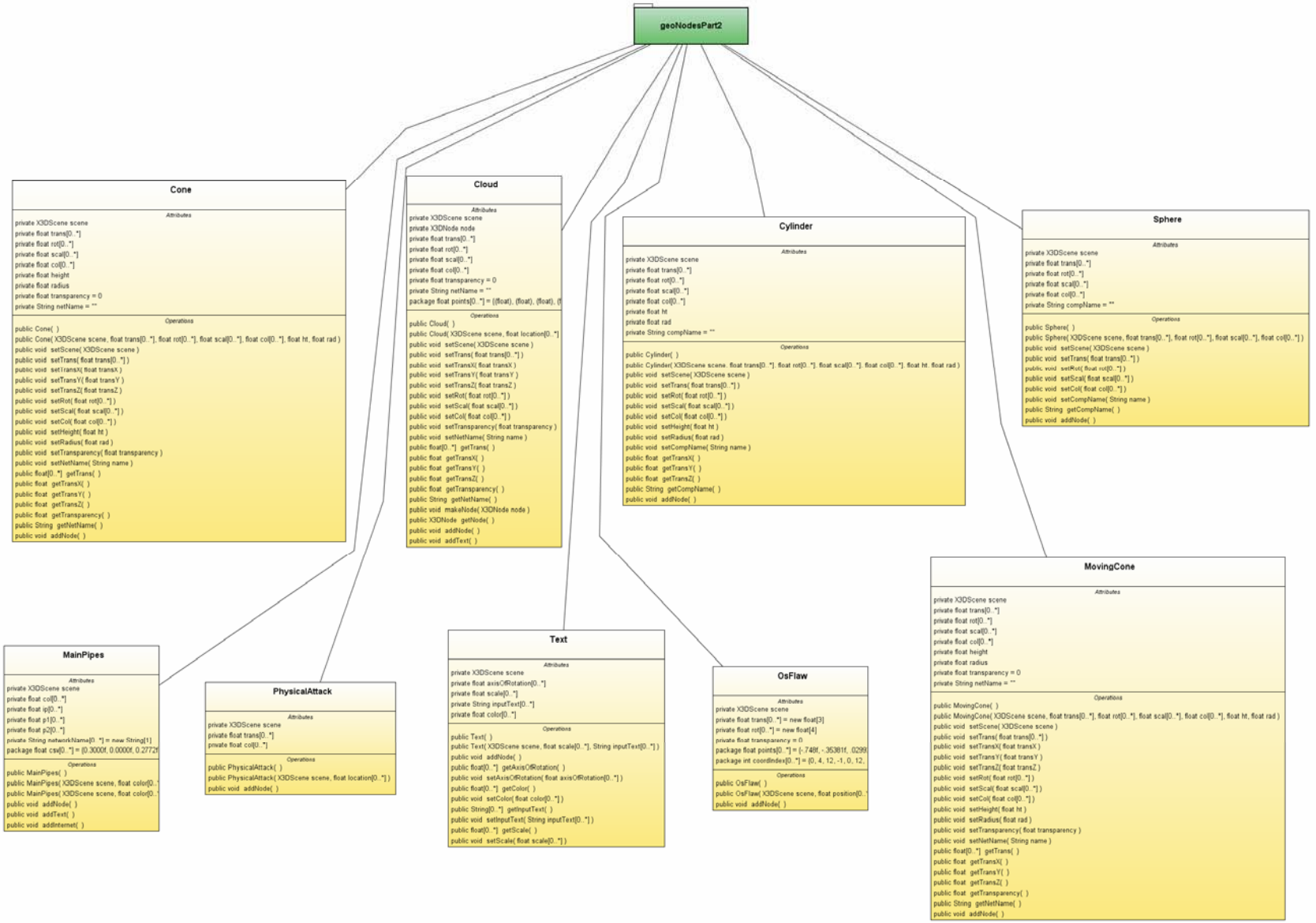
geoNodes: Contain the classes that define all of the geometric objects that appear in NTAV3D, such as computer, server, and router models, as well as the transparent walls, moving cones, etc.

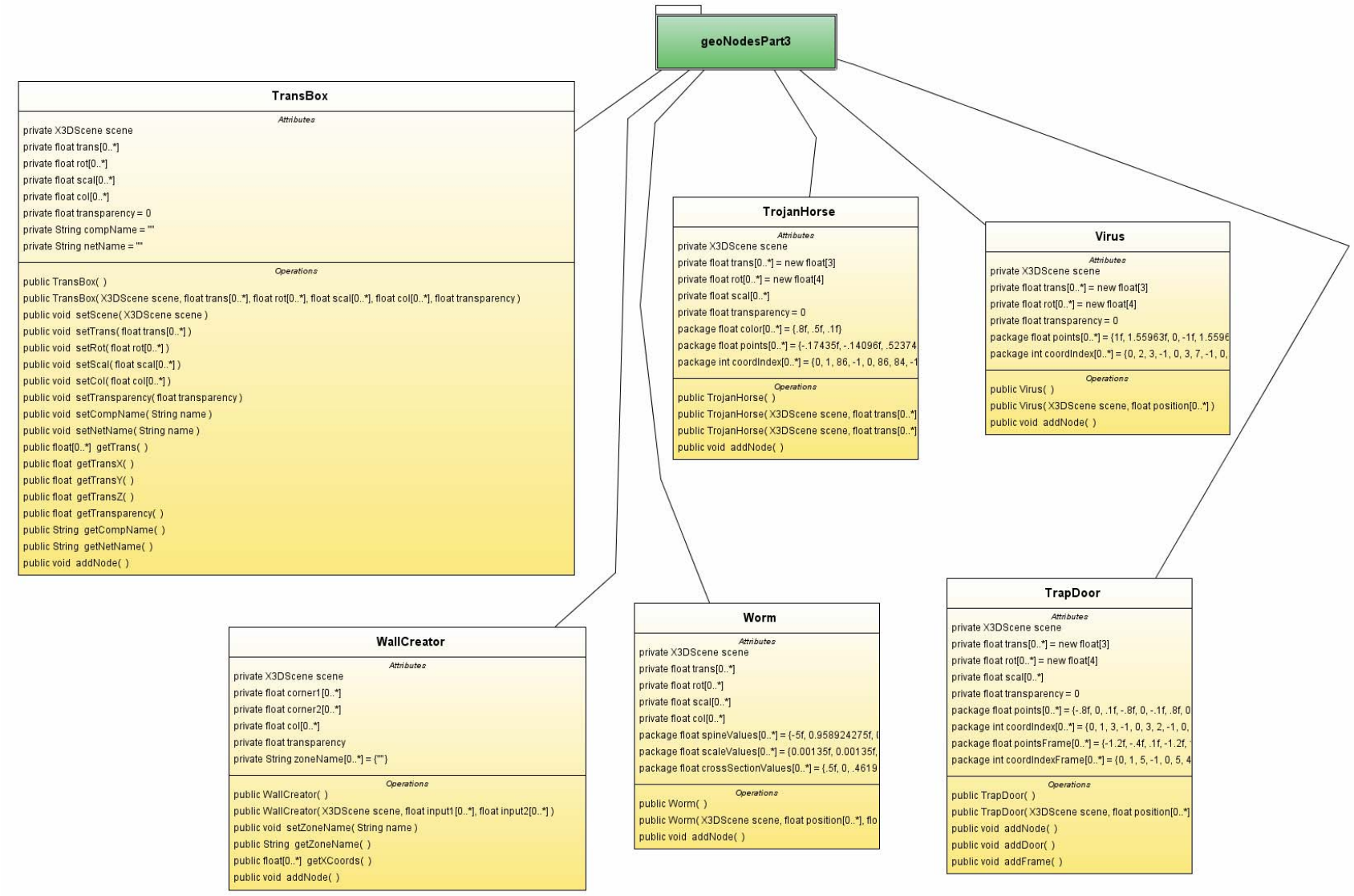
utility: Contains the main XML parser class, which carries out the bulk of the work in parsing the XML file and creating the three dimensional scene graph. The utility package also contains various utility classes, such as Java array resizers and the classes that interpret positions.

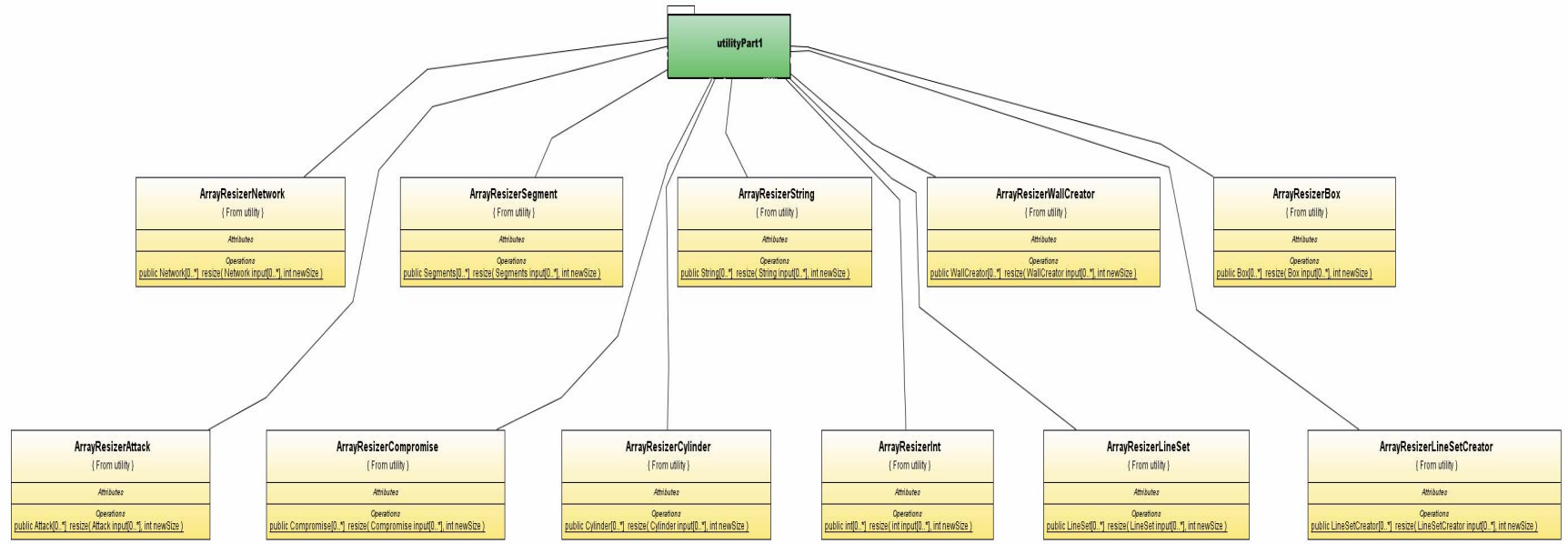

sledzCoomesThesis
SledzCoomesThesisMain
{ From sledzCoomesThesis }
Attributes
private long serialVersionUID = 385452894127295812L
private X3DScene mainScene
protected ExternalBrowser x3dBrowser
Operations
public SledzCoomesThesisMain( )
public X3DScene getMainScene( )
public void setMainScene(X3DScene mainScene)
protected ExternalBrowser getX3dBrowser( )
public void main(String args[0..*])

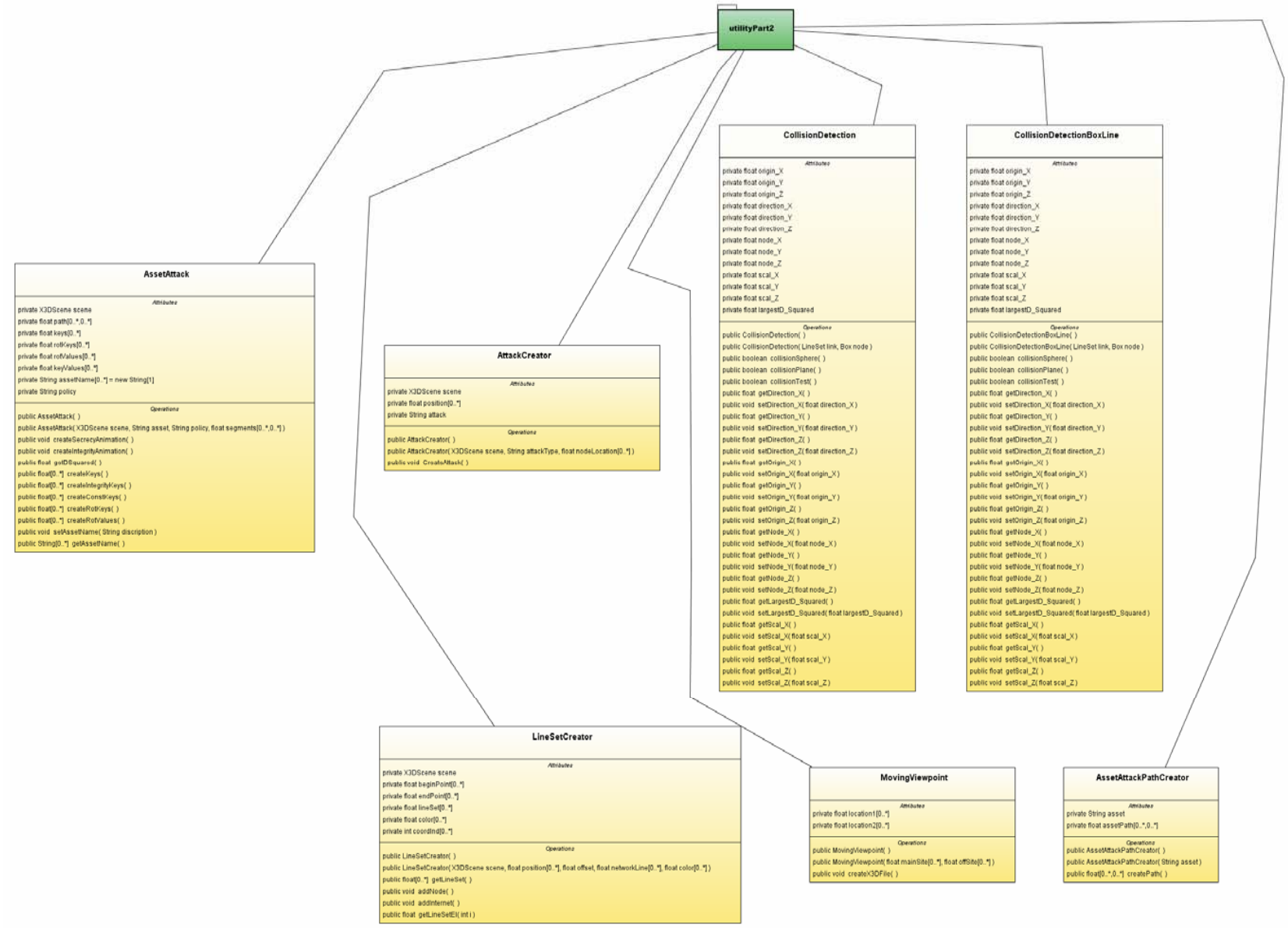

dataNodes
AttackStorage
{ From dataNodes }
Attributes
Operations
public AttackStorage( )
public Attack[0..*] getAttackArr( )
public Attack getAttackArrEl(int x)
public void setAttackArr(int hold, X3DScene scene)
public void resizeAttackArr( )
Segments
{ From dataNodes }
Attributes
private X3DScene scene
private String networkName = ""
private String componentName = ""
Operations
public Segments( )
public Segments(X3DScene scene)
public void setNetworkName(String networkName)
public void setComponentName(String componentName)
public String getComponentName( )
public String getNetworkName( )
NetworkStorage
{ From dataNodes }
Attributes
Operations
public Network[0..*] getNetworksArr( )
public Network getNetworkArrEl(int x)
public void setNetworkArr(int hold, X3DScene scene)
public void resizeNetworkArr( )
CompromiseStorage
{ From dataNodes }
Attributes
Operations
public CompromiseStorage( )
public Compromise[0..*] getCompromiseArr( )
public Compromise getCompromiseArrEl(int x)
public void setCompromiseArr(int hold, X3DScene scene)
public void resizeCompromiseArr( )
attackArr
0..*
0..*
segmentsArr
networkArr
0..*
compromiseArr
0..*
Attack
{ From dataNodes }
Attributes
private X3DScene scene
private String assetAttacked
private String policy
private boolean attackerExists
Operations
public Attack( )
public Attack(X3DScene scene)
public void setAttackerExists( )
public boolean getAttackerExists( )
public void setPolicy(String policy)
public String getPolicy( )
public void setAssetAttacked(String assetName)
public String getAssetAttacked( )
public Segments[0..*] getSegmentsArr( )
public Segments getSegmentsArrEl(int x)
public void setSegmentsArr(int hold, X3DScene scene)
public void resizeSegmentsArr( )
Network
{ From dataNodes }
Attributes
private X3DScene scene
private String networkName
private float color[0..*]
private float linep[0..*]
private String strColor = ""
Operations
public Network( )
public Network(X3DScene scene)
public String getNetworkName( )
public void setNetworkName(String networkName)
public float[0..*] getColor( )
public String getStrColor( )
public void setColor(String input)
public float[0..*] getLinep( )
public float getLinepEl(int i)
public void setLinep(float linep[0..*])
public void addNode( )
public void addInternet(float cloudPos[0..*])
Compromise
{ From dataNodes }
Attributes
private X3DScene scene
private String attackType = ""
private String attackedNode = ""
Operations
public Compromise( )
public Compromise(X3DScene scene)
public void setAttackType(String attackType)
public String getAttackType( )
public void setAttackedNode(String attackedNode)
public String getAttackedNode( )

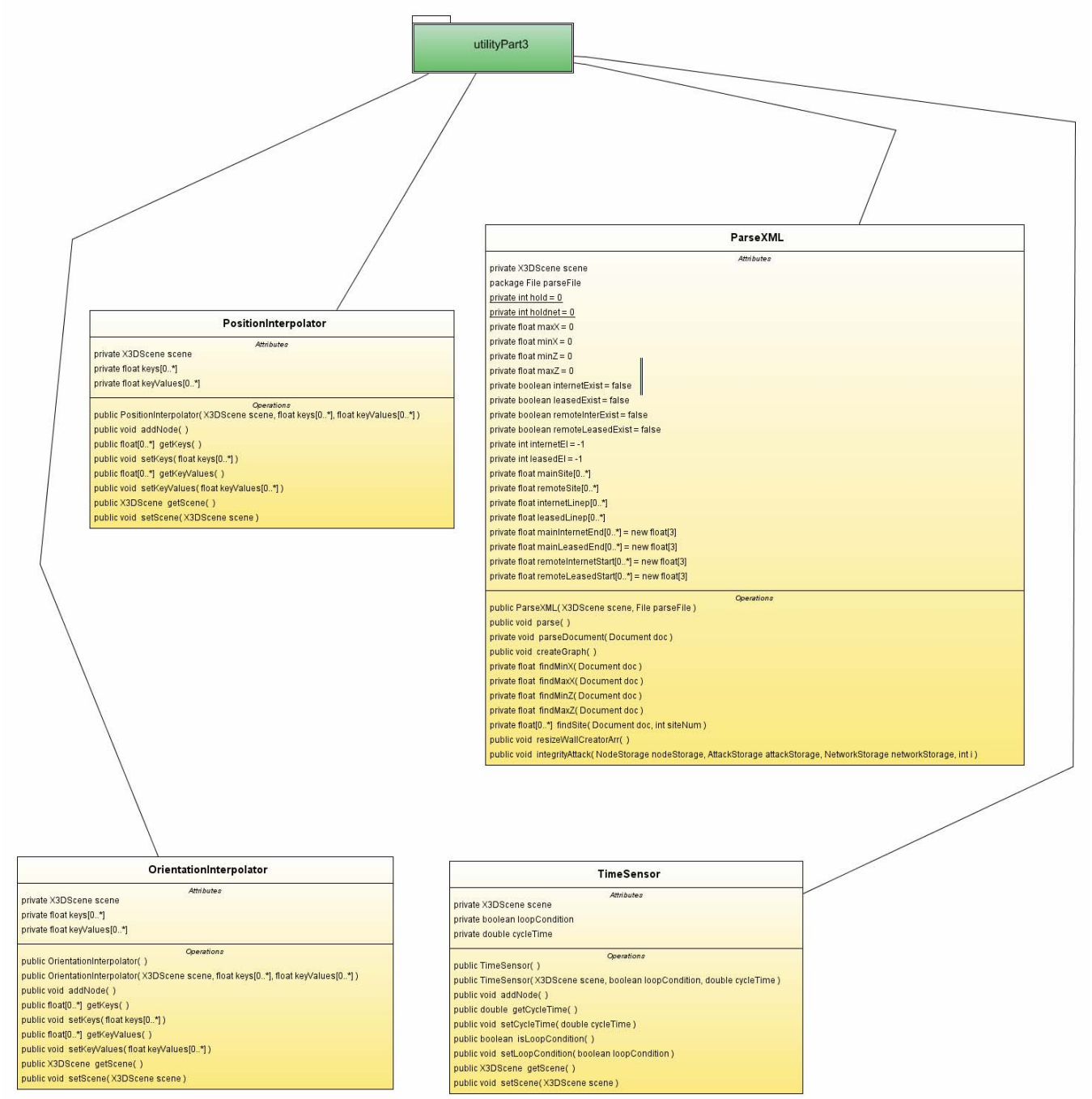

geoNodesPart1
AttackerIcon
Attributes
private X3DScene scene
private float trans[0..*] = new float[3]
private float rot[0..*] = new float[4]
private float transparency = 0
package float points[0..*] = {1f, -1.2f, 1f, 1f, - 7359
package int coordIndex[0..*] = {12, 52, 13, -1, 13, (
package float textureCoordinates[0..*] = {1f, 0f, 1f,
package int textCoordIndex[0..*] = {12, 52, 13, -1,
Operations
public AttackerIcon( )
public AttackerIcon(X3DScene scene, float positio
public void addNode( )
NodeStorage
Attributes
Operations
public NodeStorage( )
public void setBoxArr(int hold, X3DScene scene, float trans[0..*], float rot[0..*], float scal[0..*], float col[0..*])
public void setBoxArr(int hold)
public void setLineArr(int hold, X3DScene scene, int coordInd[0..*], float point[0..*], float color[0..*])
public Box[0..*] getBoxArr( )
public LineSet[0..*] getLineArr( )
public Box getBoxArrEl(int x)
public LineSet getLineArrEl(int x)
public void resizeBoxArr( )
public void resizeLineArr( )
TouchSensorText
Attributes
private X3DScene scene
private float trans[0..*]
private float rot[0..*]
private float scal[0..*]
private float col[0..*]
private float height
private float radius
Operations
public TouchSensorText( )
public TouchSensorText(X3DScene scene, float trans[0..*])
public void addNode( )
Cloud
Attributes
private X3DScene scene
private X3DNode node
private float trans[0..*]
private float rot[0..*]
private float scal[0..*]
private float col[0..*]
private float transparency = 0
private String netName = ""
package float points[0..*] = {(float), (float), (float), (f
Operations
public Cloud( )
public Cloud(X3DScene scene, float location[0..*]
public void setScene(X3DScene scene)
public void setTrans(float trans[0..*])
public void setTransX(float transX)
public void setTransY(float transY)
public void setTransZ(float transZ)
public void setRot(float rot[0..*])
public void setScal(float scal[0..*])
public void setCol(float col[0..*])
public void setTransparency(float transparency)
public void setNetName(String name)
public float[0..*] getTrans( )
public float getTransX( )
public float getTransY( )
public float getTransZ( )
public float getTransparency( )
public String getNetName( )
public void makeNode(X3DNode node)
public X3DNode getNode( )
public void addNode( )
public void addText( )
boxArr
0..*
lineArr
0..*
LineSet
Attributes
private X3DScene scene
private int coordInd[0..*]
private float point[0..*]
private float color[0..*]
private String compName = ""
private String netName = ""
Operations
public LineSet( )
public LineSet(X3DScene scene, int coordInd[0..*], float point[0..*], float color[0..*])
public void setScene(X3DScene scene)
public void setCoord(int coordInd[0..*])
public void setPoint(float point[0..*])
public void setColor(float color[0..*])
public void setCompName(String name)
public void setNetName(String name)
public int[0..*] getCoordInd( )
public String getCompName( )
public String getNetName( )
public float[0..*] getPoint( )
public float[0..*] getColor( )
public void addNode( )
Box
Attributes
private X3DScene scene
private float trans[0..*]
private float axisOfRotation[0..*]
private String inputText[0..*]
private float rot[0..*]
private float scal[0..*]
private float col[0..*]
private float transparency = 0
private int SERVER = 1
private int ROUTER = 2
private int COMPUTER = 3
private int compType = 0
private String compName = ""
private String compBase = ""
private String asset = ""
private String networks[0..*] = new String[1]
private String zoneName = ""
private String lineToNetwork
Operations
public Box( )
public Box(X3DScene scene, float trans[0..*], float rot[0..*], float scal[0..*], float col[0..*])
public void setScene(X3DScene scene)
public void setTrans(float trans[0..*])
public void setTransX(float transX)
public void setTransY(float transY)
public void setTransZ(float transZ)
public void setRot(float rot[0..*])
public void setScal(float scal[0..*])
public void setCol(float col[0..*])
public void setTransparency(float transparency)
public void setAsset(String asset)
public String getAsset( )
public void setCompType(int compType)
public void setCompName(String name)
public String getCompBase( )
public void setCompBase(String compBase)
public String[0..*] getNetworks( )
public String getNetworks(int x)
public void setNetworks(int x, String networks)
public Network[0..*] getNetworks2( )
public Network getNetworks2(int x)
public void setNetworks2(int x, Network networks2)
public String getZoneName( )
public void setZoneName(String zoneName)
public int getCompType( )
public float[0..*] getTrans( )
public float getTransX( )
public float getTransY( )
public float getTransZ( )
public float getTransparency( )
public String getCompName( )
public LineSetCreator getLine( )
public void setLine(X3DScene scene, float position[0..*], float offset, float networkLine[0..*], float color[0..*])
public LineSetCreator getLineSetEl(int i)
public void setLineSetEl(int i)
public void setLineToNetwork(String networkName)
public String getLineToNetwork( )
public void setText(String text)
public void addText( )
public float getScal(int i)
public void addNode( )
public void resizeStringArr( )
public void resizeNetworksArr( )
public void resizeLineSetCreatorArr( )


geoNodesPart2
Cone
Attributes
private X3DScene scene
private float trans[0..*]
private float rot[0..*]
private float scal[0..*]
private float col[0..*]
private float height
private float radius
private float transparency = 0
private String netName = ""
Operations
public Cone( )
public Cone(X3DScene scene, float trans[0..*], float rot[0..*], float scal[0..*], float col[0..*], float ht, float rad)
public void setScene(X3DScene scene)
public void setTrans(float trans[0..*])
public void setTransX(float transX)
public void setTransY(float transY)
public void setTransZ(float transZ)
public void setRot(float rot[0..*])
public void setScal(float scal[0..*])
public void setCol(float col[0..*])
public void setHeight(float ht)
public void setRadius(float rad)
public void setTransparency(float transparency)
public void setNetName(String name)
public float[0..*] getTrans( )
public float getTransX( )
public float getTransY( )
public float getTransZ( )
public float getTransparency( )
public String getNetName( )
public void addNode( )
Cloud
Attributes
private X3DScene scene
private X3DNode node
private float trans[0..*]
private float rot[0..*]
private float scal[0..*]
private float col[0..*]
private float transparency = 0
private String netName = ""
package float points[0..*] = {(float), (float), (float), (f
Operations
public Cloud( )
public Cloud(X3DScene scene, float location[0..*])
public void setScene(X3DScene scene)
public void setTrans(float trans[0..*])
public void setTransX(float transX)
public void setTransY(float transY)
public void setTransZ(float transZ)
public void setRot(float rot[0..*])
public void setScal(float scal[0..*])
public void setCol(float col[0..*])
public void setTransparency(float transparency)
public void setNetName(String name)
public float[0..*] getTrans( )
public float getTransX( )
public float getTransY( )
public float getTransZ( )
public float getTransparency( )
public String getNetName( )
public void makeNode(X3DNode node)
public X3DNode getNode( )
public void addNode( )
public void addText( )
Cylinder
Attributes
private X3DScene scene
private float trans[0..*]
private float rot[0..*]
private float scal[0..*]
private float col[0..*]
private float ht
private float rad
private String compName = ""
Operations
public Cylinder( )
public Cylinder(X3DScene scene, float trans[0..*], float rot[0..*], float scal[0..*], float col[0..*], float ht, float rad)
public void setScene(X3DScene scene)
public void setTrans(float trans[0..*])
public void setRot(float rot[0..*])
public void setScal(float scal[0..*])
public void setCol(float col[0..*])
public void setHeight(float ht)
public void setRadius(float rad)
public void setCompName(String name)
public float getTransX( )
public float getTransY( )
public float getTransZ( )
public String getCompName( )
public void addNode( )
Sphere
Attributes
private X3DScene scene
private float trans[0..*]
private float rot[0..*]
private float scal[0..*]
private float col[0..*]
private String compName = ""
Operations
public Sphere( )
public Sphere(X3DScene scene, float trans[0..*], float rot[0..*], float scal[0..*], float col[0..*])
public void setScene(X3DScene scene)
public void setTrans(float trans[0..*])
public void setRot(float rot[0..*])
public void setScal(float scal[0..*])
public void setCol(float col[0..*])
public void setCompName(String name)
public String getCompName( )
public void addNode( )
MainPipes
Attributes
private X3DScene scene
private float col[0..*]
private float ip[0..*]
private float p1[0..*]
private float p2[0..*]
private String networkName[0..*] = new String[1]
package float csv[0..*] = {0.3000f, 0.0000f, 0.2772f
Operations
public MainPipes( )
public MainPipes(X3DScene scene, float color[0..
public MainPipes(X3DScene scene, float color[0..
public void addNode( )
public void addText( )
public void addInternet( )
PhysicalAttack
Attributes
private X3DScene scene
private float trans[0..*]
private float col[0..*]
Operations
public PhysicalAttack( )
public PhysicalAttack(X3DScene scene, float location[0..*])
public void addNode( )
Text
Attributes
private X3DScene scene
private float axisOfRotation[0..*]
private float scale[0..*]
private String inputText[0..*]
private float color[0..*]
Operations
public Text( )
public Text(X3DScene scene, float scale[0..*], String inputText[0..*])
public void addNode( )
public float[0..*] getAxisOfRotation( )
public void setAxisOfRotation(float axisOfRotation[0..*])
public float[0..*] getColor( )
public void setColor(float color[0..*])
public String[0..*] getInputText( )
public void setInputText(String inputText[0..*])
public float[0..*] getScale( )
public void setScale(float scale[0..*])
OsFlaw
Attributes
private X3DScene scene
private float trans[0..*] = new float[3]
private float rot[0..*] = new float[4]
private float transparency = 0
package float points[0..*] = {-.748f, -.35381f, .0299
package int coordIndex[0..*] = {0, 4, 12, -1, 0, 12,
Operations
public OsFlaw( )
public OsFlaw(X3DScene scene, float position[0..
public void addNode( )
MovingCone
Attributes
private X3DScene scene
private float trans[0..*]
private float rot[0..*]
private float scal[0..*]
private float col[0..*]
private float height
private float radius
private float transparency = 0
private String netName = ""
Operations
public MovingCone( )
public MovingCone(X3DScene scene, float trans[0..*], float rot[0..*], float scal[0..*], float col[0..*], float ht, float rad)
public void setScene(X3DScene scene)
public void setTrans(float trans[0..*])
public void setTransX(float transX)
public void setTransY(float transY)
public void setTransZ(float transZ)
public void setRot(float rot[0..*])
public void setScal(float scal[0..*])
public void setCol(float col[0..*])
public void setHeight(float ht)
public void setRadius(float rad)
public void setTransparency(float transparency)
public void setNetName(String name)
public float[0..*] getTrans( )
public float getTransX( )
public float getTransY( )
public float getTransZ( )
public float getTransparency( )
public String getNetName( )
public void addNode( )


geoNodesPart3
TransBox
Attributes
private X3DScene scene
private float trans[0..*]
private float rot[0..*]
private float scal[0..*]
private float col[0..*]
private float transparency = 0
private String compName = ""
private String netName = ""
Operations
public TransBox( )
public TransBox(X3DScene scene, float trans[0..*], float rot[0..*], float scal[0..*], float col[0..*], float transparency)
public void setScene(X3DScene scene)
public void setTrans(float trans[0..*])
public void setRot(float rot[0..*])
public void setScal(float scal[0..*])
public void setCol(float col[0..*])
public void setTransparency(float transparency)
public void setCompName(String name)
public void setNetName(String name)
public float[0..*] getTrans( )
public float getTransX( )
public float getTransY( )
public float getTransZ( )
public float getTransparency( )
public String getCompName( )
public String getNetName( )
public void addNode( )
TrojanHorse
Attributes
private X3DScene scene
private float trans[0..*] = new float[3]
private float rot[0..*] = new float[4]
private float scal[0..*]
private float transparency = 0
package float color[0..*] = {.8f, .5f, .1f}
package float points[0..*] = {-.17435f, -.14096f, .52374
package int coordIndex[0..*] = {0, 1, 86, -1, 0, 86, 84, -1
Operations
public TrojanHorse( )
public TrojanHorse(X3DScene scene, float trans[0..*]
public TrojanHorse(X3DScene scene, float trans[0..*]
public void addNode( )
Virus
Attributes
private X3DScene scene
private float trans[0..*] = new float[3]
private float rot[0..*] = new float[4]
private float transparency = 0
package float points[0..*] = {1f, 1.55963f, 0, -1f, 1.5596
package int coordIndex[0..*] = {0, 2, 3, -1, 0, 3, 7, -1, 0,
Operations
public Virus( )
public Virus(X3DScene scene, float position[0..*])
public void addNode( )
WallCreator
Attributes
private X3DScene scene
private float corner1[0..*]
private float corner2[0..*]
private float col[0..*]
private float transparency
private String zoneName[0..*] = {""}
Operations
public WallCreator( )
public WallCreator(X3DScene scene, float input1[0..*], float input2[0..*])
public void setZoneName(String name)
public String getZoneName( )
public float[0..*] getXCoords( )
public void addNode( )
Worm
Attributes
private X3DScene scene
private float trans[0..*]
private float rot[0..*]
private float scal[0..*]
private float col[0..*]
package float spineValues[0..*] = {-5f, 0.958924275f,
package float scaleValues[0..*] = {0.00135f, 0.00135f,
package float crossSectionValues[0..*] = {.5f, 0, .4619
Operations
public Worm( )
public Worm(X3DScene scene, float position[0..*], flo
public void addNode( )
TrapDoor
Attributes
private X3DScene scene
private float trans[0..*] = new float[3]
private float rot[0..*] = new float[4]
private float scal[0..*]
private float transparency = 0
package float points[0..*] = {-.8f, 0, .1f, -.8f, 0, -.1f, .8f, 0
package int coordIndex[0..*] = {0, 1, 3, -1, 0, 3, 2, -1, 0,
package float pointsFrame[0..*] = {-1.2f, -.4f, .1f, -1.2f,
package int coordIndexFrame[0..*] = {0, 1, 5, -1, 0, 5, 4
Operations
public TrapDoor( )
public TrapDoor(X3DScene scene, float position[0..*]
public void addNode( )
public void addDoor( )
public void addFrame( )


utilityPart1
ArrayResizerNetwork
{ From utility }
Attributes
Operations
public Network[0..*] resize(Network input[0..*], int newSize)
ArrayResizerSegment
{ From utility }
Attributes
Operations
public Segments[0..*] resize(Segments input[0..*], int newSize)
ArrayResizerString
{ From utility }
Attributes
Operations
public String[0..*] resize(String input[0..*], int newSize)
ArrayResizerWallCreator
{ From utility }
Attributes
Operations
public WallCreator[0..*] resize(WallCreator input[0..*], int newSize)
ArrayResizerBox
{ From utility }
Attributes
Operations
public Box[0..*] resize(Box input[0..*], int newSize)
ArrayResizerAttack
{ From utility }
Attributes
Operations
public Attack[0..*] resize(Attack input[0..*], int newSize)
ArrayResizerCompromise
{ From utility }
Attributes
Operations
public Compromise[0..*] resize(Compromise input[0..*], int newSize)
ArrayResizerCylinder
{ From utility }
Attributes
Operations
public Cylinder[0..*] resize(Cylinder input[0..*], int newSize)
ArrayResizerInt
{ From utility }
Attributes
Operations
public int[0..*] resize(int input[0..*], int newSize)
ArrayResizerLineSet
{ From utility }
Attributes
Operations
public LineSet[0..*] resize(LineSet input[0..*], int newSize)
ArrayResizerLineSetCreator
{ From utility }
Attributes
Operations
public LineSetCreator[0..*] resize(LineSetCreator input[0..*], int newSize)


utilityPart2
AssetAttack
Attributes
private X3DScene scene
private float path[0..*,0..*]
private float keys[0..*]
private float rotKeys[0..*]
private float rotValues[0..*]
private float keyValues[0..*]
private String assetName[0..*] = new String[1]
private String policy
Operations
public AssetAttack( )
public AssetAttack(X3DScene scene, String asset, String policy, float segments[0..*,0..*])
public void createSecrecyAnimation( )
public void createIntegrityAnimation( )
public float getDSquared( )
public float[0..*] createKeys( )
public float[0..*] createIntegrityKeys( )
public float[0..*] createConstKeys( )
public float[0..*] createRotKeys( )
public float[0..*] createRotValues( )
public void setAssetName(String discription)
public String[0..*] getAssetName( )
AttackCreator
Attributes
private X3DScene scene
private float position[0..*]
private String attack
Operations
public AttackCreator( )
public AttackCreator(X3DScene scene, String attackType, float nodeLocation[0..*])
public void CreateAttack( )
CollisionDetection
Attributes
private float origin_X
private float origin_Y
private float origin_Z
private float direction_X
private float direction_Y
private float direction_Z
private float node_X
private float node_Y
private float node_Z
private float scal_X
private float scal_Y
private float scal_Z
private float largestD_Squared
Operations
public CollisionDetection( )
public CollisionDetection(LineSet link, Box node)
public boolean collisionSphere( )
public boolean collisionPlane( )
public boolean collisionTest( )
public float getDirection_X( )
public void setDirection_X(float direction_X)
public float getDirection_Y( )
public void setDirection_Y(float direction_Y)
public float getDirection_Z( )
public void setDirection_Z(float direction_Z)
public float getOrigin_X( )
public void setOrigin_X(float origin_X)
public float getOrigin_Y( )
public void setOrigin_Y(float origin_Y)
public float getOrigin_Z( )
public void setOrigin_Z(float origin_Z)
public float getNode_X( )
public void setNode_X(float node_X)
public float getNode_Y( )
public void setNode_Y(float node_Y)
public float getNode_Z( )
public void setNode_Z(float node_Z)
public float getLargestD_Squared( )
public void setLargestD_Squared(float largestD_Squared)
public float getScal_X( )
public void setScal_X(float scal_X)
public float getScal_Y( )
public void setScal_Y(float scal_Y)
public float getScal_Z( )
public void setScal_Z(float scal_Z)
CollisionDetectionBoxLine
Attributes
private float origin_X
private float origin_Y
private float origin_Z
private float direction_X
private float direction_Y
private float direction_Z
private float node_X
private float node_Y
private float node_Z
private float scal_X
private float scal_Y
private float scal_Z
private float largestD_Squared
Operations
public CollisionDetectionBoxLine( )
public CollisionDetectionBoxLine(LineSet link, Box node)
public boolean collisionSphere( )
public boolean collisionPlane( )
public boolean collisionTest( )
public float getDirection_X( )
public void setDirection_X(float direction_X)
public float getDirection_Y( )
public void setDirection_Y(float direction_Y)
public float getDirection_Z( )
public void setDirection_Z(float direction_Z)
public float getOrigin_X( )
public void setOrigin_X(float origin_X)
public float getOrigin_Y( )
public void setOrigin_Y(float origin_Y)
public float getOrigin_Z( )
public void setOrigin_Z(float origin_Z)
public float getNode_X( )
public void setNode_X(float node_X)
public float getNode_Y( )
public void setNode_Y(float node_Y)
public float getNode_Z( )
public void setNode_Z(float node_Z)
public float getLargestD_Squared( )
public void setLargestD_Squared(float largestD_Squared)
public float getScal_X( )
public void setScal_X(float scal_X)
public float getScal_Y( )
public void setScal_Y(float scal_Y)
public float getScal_Z( )
public void setScal_Z(float scal_Z)
LineSetCreator
Attributes
private X3DScene scene
private float beginPoint[0..*]
private float endPoint[0..*]
private float lineSet[0..*]
private float color[0..*]
private int coordInd[0..*]
Operations
public LineSetCreator( )
public LineSetCreator(X3DScene scene, float position[0..*], float offset, float networkLine[0..*], float color[0..*])
public float[0..*] getLineSet( )
public void addNode( )
public void addInternet( )
public float getLineSetEl(int i)
MovingViewpoint
Attributes
private float location1[0..*]
private float location2[0..*]
Operations
public MovingViewpoint( )
public MovingViewpoint(float mainSite[0..*], float offSite[0..*])
public void createX3DFile( )
AssetAttackPathCreator
Attributes
private String asset
private float assetPath[0..*,0..*]
Operations
public AssetAttackPathCreator( )
public AssetAttackPathCreator(String asset)
public float[0..*,0..*] createPath( )


utilityPart3
PositionInterpolator
Attributes
private X3DScene scene
private float keys[0..*]
private float keyValues[0..*]
Operations
public PositionInterpolator(X3DScene scene, float keys[0..*], float keyValues[0..*])
public void addNode( )
public float[0..*] getKeys( )
public void setKeys(float keys[0..*])
public float[0..*] getKeyValues( )
public void setKeyValues(float keyValues[0..*])
public X3DScene getScene( )
public void setScene(X3DScene scene)
ParseXML
Attributes
private X3DScene scene
package File parseFile
private int hold = 0
private int holdnet = 0
private float maxX = 0
private float minX = 0
private float minZ = 0
private float maxZ = 0
private boolean internetExist = false
private boolean leasedExist = false
private boolean remoteInterExist = false
private boolean remoteLeasedExist = false
private int internetEl = -1
private int leasedEl = -1
private float mainSite[0..*]
private float remoteSite[0..*]
private float internetLinep[0..*]
private float leasedLinep[0..*]
private float mainInternetEnd[0..*] = new float[3]
private float mainLeasedEnd[0..*] = new float[3]
private float remoteInternetStart[0..*] = new float[3]
private float remoteLeasedStart[0..*] = new float[3]
Operations
public ParseXML(X3DScene scene, File parseFile)
public void parse( )
private void parseDocument(Document doc)
public void createGraph( )
private float findMinX(Document doc)
private float findMaxX(Document doc)
private float findMinZ(Document doc)
private float findMaxZ(Document doc)
private float[0..*] findSite(Document doc, int siteNum)
public void resizeWallCreatorArr( )
public void integrityAttack(NodeStorage nodeStorage, AttackStorage attackStorage, NetworkStorage networkStorage, int i)
OrientationInterpolator
Attributes
private X3DScene scene
private float keys[0..*]
private float keyValues[0..*]
Operations
public OrientationInterpolator( )
public OrientationInterpolator(X3DScene scene, float keys[0..*], float keyValues[0..*])
public void addNode( )
public float[0..*] getKeys( )
public void setKeys(float keys[0..*])
public float[0..*] getKeyValues( )
public void setKeyValues(float keyValues[0..*])
public X3DScene getScene( )
public void setScene(X3DScene scene)
TimeSensor
Attributes
private X3DScene scene
private boolean loopCondition
private double cycleTime
Operations
public TimeSensor( )
public TimeSensor(X3DScene scene, boolean loopCondition, double cycleTime)
public void addNode( )
public double getCycleTime( )
public void setCycleTime(double cycleTime)
public boolean isLoopCondition( )
public void setLoopCondition(boolean loopCondition)
public X3DScene getScene( )
public void setScene(X3DScene scene)

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. SAMPLE XML FILES

A. CYBERCIEGE XML DOCUMENT TYPE DEFINITION (NETVIEW.DTD)

```
<!--
   Name:    netView.dtd
   Version: Draft 1.2
-->
<!ELEMENT CyberCIEGEnetView ((version, SDFid, network+, mainSite, offSite?, componentCompromise*, attack*))>
<!ELEMENT version (#PCDATA)>
<!ELEMENT SDFid (#PCDATA)>
<!--
        Associate colors with network names to match that seen in game.
-->
<!ELEMENT network (networkName, color)>
<!ELEMENT mainSite (zoneName, upperLeft, lowerRight, component*, zone*)>
<!ELEMENT offSite (zoneName, upperLeft, lowerRight, component*)>
<!--
        zones can be nested.
-->
<!ELEMENT zone (zoneName, upperLeft, lowerRight, component*, zone*)>
<!--
        In the future component may include O/S name and configuration settings.  In initial implementation, assets need not be depicted on other than the computer that is attacked.
-->
<!ELEMENT component (componentName, componentBase, location, networkName*, assetName*)>
<!--
        The component name as assigned by player or engine, e.g., "Joe's PC"
-->
<!ELEMENT componentName (#PCDATA)>
<!--
        Identifies the graphics image to use when representing this component
-->
<!ELEMENT componentBase (#PCDATA)>
<!--
        Dimensions are between 0 and 100 on each axis. Component locations will all tend to have the same "y" component, except when located in a server rack.  For now, those components floating above each other will be sufficient (i.e., no need to illustrate a rack).
-->
<!ELEMENT upperLeft (x, z)>
<!ELEMENT lowerRight (x, z)>
<!ELEMENT location (x, y, z)>
```

```
<!ELEMENT x (#PCDATA)>
<!ELEMENT y (#PCDATA)>
<!ELEMENT z (#PCDATA)>
<!ELEMENT networkName (#PCDATA)>
<!ELEMENT assetName (#PCDATA)>
<!--
        The componentCompromise pairs identify which components should include
        which attack type visual representation.  There is currently no semantic linkage
        between componentCompromise and attacks.
-->
<!ELEMENT componentCompromise (attackType, componentName)>
<!ELEMENT attackType (#PCDATA)>
<!--<!ELEMENT attackType (trojanHorse | trapDoor | osFlaw | physical | virus)>-->
<!--
        The attack simply illustrates the asset being compromised.  A policy of "secrecy"
        will show information flowing from the asset out throught the networks.  A policy
        of "integrity" will show information flowing from the networks into the asset.  If
        no segments are provided, the flow will simply be into and out of the component
        that contains the asset, and if "attacker" is present the flow will be to-from the
        attacker.  If attacker is not present, the flow will be internal to the computer (i.e.,
        it would be an integrity compromise driven by malware).

        The segments simply reflect the path the data takes on its way to or from the
        attacker. Segments are order dependent.  If a "attacker" is present, then the
        componentName in the final segment is the component at which  the attacker
        receives information from secrecy attacks or sends information in integrity
        attacks.  In a typical secrecy case, information would be depicted as flowing from
        the asset, through a series of networks  and components, and then out of the final
        component into an attacker icon.  If the final segment includes a  zoneName
        instead of a componentName, then the information enters or leaves the final
        network anywhere within the named zone.  If there is no named zone or
        component in the final segment then the network will be exernal (either the
        Internet or some network that goes between the main site and the offsite) - in
        which case the attacker is at the Internet or the other network link.
-->
<!ELEMENT attack (assetName, policy, segment*, attacker?)>
<!ELEMENT segment (Network, (componentName? | zoneName?))>
<!ELEMENT Network (#PCDATA)>
<!ELEMENT policy (#PCDATA)>
<!--<!ELEMENT policy (Secrecy | Integrity)>-->
<!ELEMENT attacker (#PCDATA)>
<!ELEMENT zoneName (#PCDATA)>
<!--
        Color is rgb in hex, e.g., 0xFF00FF00
-->
<!ELEMENT color (#PCDATA)>
```

B. SAMPLE CYBERCIEGE XML OUTPUT

```
<?xml version="1.0"?>
<!DOCTYPE CyberCIEGEnetView SYSTEM "netView.dtd">
<CyberCIEGEnetView>
        <version>0.0</version>
        <SDFid>0.0</SDFid>
        <network>
                <networkName>Internet</networkName>
                <color>0xFFFF0000</color>
        </network>
        <network>
                <networkName>leased</networkName>
                <color>0xFF00FF00</color>
        </network>
        <network>
                <networkName>link1</networkName>
                <color>0xFFFF7F00</color>
        </network>
        <network>
                <networkName>link2</networkName>
                <color>0xFF0000FF</color>
        </network>
        <network>
                <networkName>Lan1</networkName>
                <color>0xFFFF00FF</color>
        </network>
        <network>
                <networkName>Offsite LAN</networkName>
                <color>0xFFFFFF00</color>
        </network>
        <mainSite>
                <zoneName>Entire Office</zoneName>
                <upperLeft><x>33</x><z>49</z></upperLeft>
                <lowerRight><x>56</x><z>31</z></lowerRight>
                <component>
                        <componentName>Joe_ws</componentName>
                        <componentBase>Blato Desktop Select</componentBase>
                        <location><x>35</x><y>1</y><z>36</z></location>
                        <networkName>leased</networkName>
                        <networkName>Lan1</networkName>
                        <assetName>Plans</assetName>
                </component>
                <component>
                        <componentName>Lunitos AFOS_3</componentName>
                        <componentBase>Lunitos AFOS</componentBase>
```

<location><x>35</x><y>0</y><z>47</z></location>
<networkName>leased</networkName>
<networkName>link1</networkName>
</component>
<component>
<componentName>Bit Flipper Border_3</componentName>
<componentBase>Bit Flipper Border</componentBase>
<location><x>34</x><y>1</y><z>37</z></location>
<networkName>Internet</networkName>
<networkName>link1</networkName>
<networkName>Lan1</networkName>
</component>
</mainSite>
<offSite>
<zoneName>Offsite</zoneName>
<upperLeft><x>94</x><z>28</z></upperLeft>
<lowerRight><x>106</x><z>21</z></lowerRight>
<component>
<componentName>Kim's Workstation</componentName>
<componentBase>Blato Desktop Select</componentBase>
<location><x>104</x><y>1</y><z>23</z></location>
<networkName>leased</networkName>
</component>
<component>
<componentName>Lunitos AFOS_2</componentName>
<componentBase>Lunitos AFOS</componentBase>
<location><x>96</x><y>1</y><z>22</z></location>
<networkName>Offsite LAN</networkName>
</component>
<component>
<componentName>Bit Flipper_2</componentName>
<componentBase>Bit Flipper</componentBase>
<location><x>95</x><y>1</y><z>23</z></location>
<networkName>Internet</networkName>
<networkName>Offsite LAN</networkName>
</component>
</offSite>
</CyberCIEGEnetView>

LIST OF REFERENCES

Alpern, B. Amd Carter, L. 1991. *HyberBox*. IEEE, Proceedings Visualisation 1991, pp. 133-139.

Au, S., Leckie, C., Parhar, A., Wong, G. *Efficient Visualization of Large Routing Topologies*. International Journal of Network Management. 2004.

Baba, T., Matsuda, S. *Tracing Network Attacks to Their Sources*. IEEE Internet Computing. March/April 2002. pp. 20-26.

Baxley, T., Xu, J., Yu, H., Zhang, J., Yuan, X., Brickhouse, J. *LAN Attacker: A Visual Education Tool*. Proceedings 3rd Annual Conference on Information Security Curriculum Development. ACM 2006.

Becker, R.A., Eick, S.G., Miller, E.O., Wilks, A.R. 1990. *Dynamic Graphics for Network Visualization*. The Institute of Electrical and Electronic Engineers, INC.

Becker, R. A., Eick, S.G., Wilks, A.R. 1995. *Visualizing Network Data*. IEEE Transactions on Visualization and Computer Graphics. Vol. 1. pp. 16-28.

Berg, C. High-Assurance Design: Architecting Secure And Reliable Enterprise Applications. Pearson Publishing. 2005.

Cone, B. D., Thompson, M. F., Irvine, C. E. and Nguyen, T. D., *Cyber Security Training and Awareness Through Game Play*, 2006, in IFIP International Federation for Information Processing, Volume 201, Security and Privacy in Dynamic Environments, eds. Fischer-Hubner, S., Rannenberg, K., Yngstrom, L., Lindskog, S., (Boston: Springer), pp. 431-436.

Cox, K. C., Eick, S. G., He, T. *3D Geographic Network Displays*. SIGMOD Record, Vol. 25, No. 4, December 1996.

Dawkins, J., Hale, J. *A Systematic Approach to Multi-State Network Attack Analysis*. Proceedings 2nd IEEE International Information Assurance Workshop 2004.

ECMA. 1999. *ECMAScript Language Specification*. <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>. June 8, 2007.

Eick, G., 1996. *Aspects of Network Visualization*. The Institute of Electrical and Electronic Engineers, INC.

Eick, S., Wills, G. *Navigating Large Networks with Hierarchies*. IEEE 1993.

Erbacher, R. *Glyph-Based Generic Network Visualization*. Proceedings of SPIE 2002 Conference on Visualization and Data Analysis.

Estrin, D., Handley, M., Heidemann, J., McCanne, S., Xu, Y., Yu, H. *Network Visualization with NAM, the VINT Network Animator*. Computer. IEEE Computer Society Press. VOL 33, Issue 11. 2000. pp. 63-68.

ISO/IEC 19775-1:2004. 2004. Web3D Consortium. <http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification/>. April 25, 2007.

Gardner, H. Frames of Mind: The Theory of Multiple Intelligences. New York. Back Books Inc. 1985.

Hansen, C., Johnson, C. Visualization Handbook. New York. Academic Press. 2004.

Holmberg, N., Wünsche, B., Tempero, E. *A Framework for Interactive Web-Based Visualization*. Proceedings 7th Australasian User Interface Conference (AUIC) 2006.

Irvine, C.E., Thompson, M.F., and Allen, K. *CyberCIEGE: Gaming for Information Assurance*. IEEE Security and Privacy, (May/June 2005), Volume 3 Issue 3, 61-64.

Keller, P. R., M. M. Keller. 1993. *Visual Cues Practical Data Visualization*. The Institute of Electrical and Electronic Engineers, INC.

Kershenbaum, A., Murray, K. *Visualization of Network Structures*. Journal of Computing Sciences in Colleges. Vol. 21, Issue 2. Consortium for Computing Sciences in Colleges 2005. pp. 59-71.

Kim, H., Kang, I., Bahk, S. *Real-Time Visualization of Network Attacks on High-Speed Links*. IEEE Network. September/October 2004.

Labib, K., Vemuri, R. *Detecting and Visualizing Denial-of-Service and Network Probe Attacks Using Principal Component Analysis*. Proceedings 3rd Conference on Security and Network Architectures. SAR 2004.

Li, L., Liu, P., Kesdis, G. *Visual Toolkit for Network Security Experiment Specification and Data Analysis*. VizSEC. ACM 2006.

Meyer, B. *Self-organizing Graphs: A Neural Network perspective of Graph Layout*. In Proceedings of the 1998 Graph Drawing Symposium (GD'98). Montreal, Canada August 1998.

Nyarko, K., Capers, T., Scott, C., Ladeji-Osias, K. *Network Intrustion Visualization with NIVA, an Intrusion Detection Visual Analyzer with Haptic Integration*. Proceedings 10th Symposium On Haptic Interfaces for Virtual Environments & Teleoperator Systems. IEEE 2002.

Schneier, B. *Attack Trees*. 8, October 1999.

Snort.org. April 20, 2007. Sourcefire Inc. <http://www.snort.org/>. April 23, 2007.

Spence, Robert. 2001. *Information Visualization*. ACM Press, A Division of the Association for Computing Machinery , INC.

Tidwell, T., Larson, R., Fitch, K., Hale, J. *Modeling Internet Attacks* Proceedings 2001 IEEE Workshop on Information Assurance and Security. pp. 54-59.

The Xj3D Project. April 21, 2006. Web3D Consortium. <http://www.xj3d.org/>. February 12, 2007.

Web3D Consortium. *What is X3D?*. <http://www.web3d.org/about/overview>. May 15, 2007.

Xj3D Licensing information. October 16, 2005. Web3D Consortium. <http://www.xj3d.org/licenses/license.html>. May 29, 2007.

Ye, N., Emran, S., Chen, Q., Vilbet, S. *Multivariate Statisical Analysis of Audit Trails for Host-Based Intrusion Detection*. IEEE Transactions on Computers. Vol. 51, No. 7. July 2002.

Ye, N., Li, X., Chen, Q., Emran, S., Xu, M. *Probabilistic Techniques for Intrusion Detection Based on Computer Audit Data*. IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans. Vol. 31, No. 4. July 2001.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Mathias Kölsch
   Naval Postgraduate School
   Monterey, California

4. Michael Thompson
   Naval Postgraduate School
   Monterey, California

5. Cynthia Irvine
   Naval Postgraduate School
   Monterey, California

6. Don Brutzman
   Naval Postgraduate School
   Monterey, California